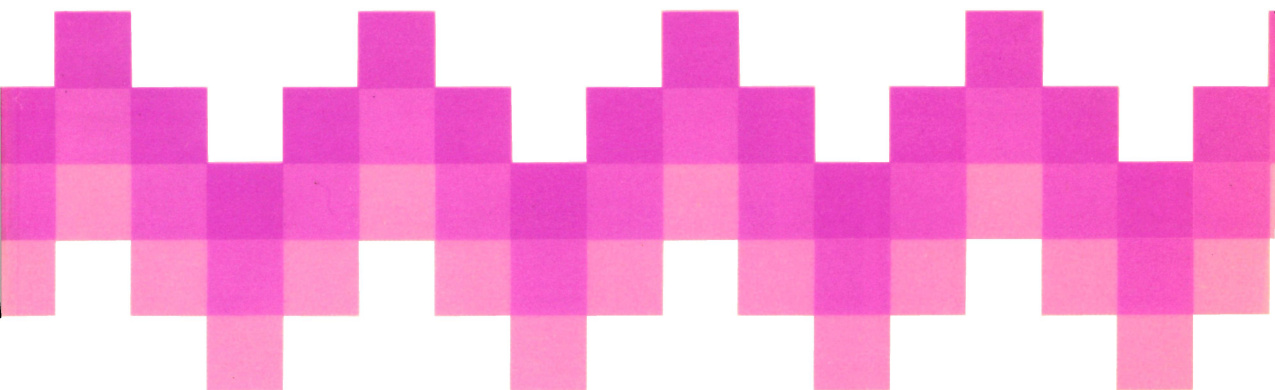


I. Quartiroli M. Fusaro S. Smareglia

UNIX

INTRODUZIONE AL SISTEMA OPERATIVO



clup

finito di stampare nel mese di ottobre 1983
presso il centro stampa rozzano, via milano 99, rozzano (mi)
per conto della clup, piazza leonardo da vinci 32, milano

UNIX is a Trademark of Bell Laboratories

copyright © 1983 clup
cooperativa libreria universitaria del politecnico, milano
prima edizione: maggio 1983
prima ristampa: ottobre 1983
ISBN 88 - 7005 - 566 - 3

**Ivo Quartiroli
Massimo Fusaro
Stefano Smareglia**

**UNIX
Introduzione al Sistema Operativo**

clup - milano

La casa editrice Clup ringrazia per la gentile collaborazione
il prof. Gianni Degli Antoni e il prof. Roberto Polillo
dell'Istituto di Cibernetica dell'Università degli Studi di Milano,
e la dott. Marta Ferrari del Laboratorio Didattico dell'Istituto.

INDICE

INTRODUZIONE	pagina	11
CAPITOLO 1 : Conoscere UNIX		15
IL FILE SYSTEM		15
File normali		15
Directory		16
File speciali		17
IL MECCANISMO DELLE PROTEZIONI		17
DA LOGIN A LOGOUT		19
PRIME ISTRUZIONI		20
Errori nello scrivere		21
Date		21
Opzioni, argomenti		22
Creazione files		22
ISTRUZIONI PER TRATTARE I FILES		25
Visualizzazione file: cat, pr, more		25
Altre istruzioni sui files		26
GLI EDITOR VI ED ED		27
Vi		28
Muoversi lungo il testo		30
Ricerche di parole		32
Inserimenti		34
Correzioni		34
Scrivere caratteri di controllo		35
Errori		36
Mettere piu' linee in una		36
Duplicazione di linee, il comando p		37
Incolonnamento di righe		37
Manipolazione di files		38
Marking		39
Gestione video		39
Ripetizioni di comandi		40
Ritrovare linee cancellate, file persi		40
Comandi per tutto il testo		41
Parametri		42
Cose che capitano in vi		43
Ed		44
Cambiamenti		45
Sostituzioni		46
Ricerche		47
Altri comandi		48
USO DI DIRECTORY		50
METACARATTERI		51
VISIONE DEL FILE-SYSTEM		53

COMUNICAZIONI TRA UTENTI	54
RIDIREZIONI	56
STAMPANTE	58
PIPE	58
PROCESSI	59
PROTEGGERE I PROPRI FILES	62
IL COMPILATORE PASCAL PI	63
Primo programma	64
Un programma piu' grosso	66
Profilazione di esecuzione	72
Errori del compilatore	74
Errori in esecuzione	74
Loop infiniti	76
Eoln ed eof	76
Reset e rewrite	77
Argc e argv	78
Tipi standard, limiti	81
Opzioni di pi	81
Pxp	82
Cross-reference	82
Procedure predefinite	82
Funzioni predefinite	84
 CAPITOLO 2 : Qualcosa di piu'	 85
I LINK	85
I FLAG DEI FILE	87
LA SHELL	89
I file di inizializzazione	89
Variabili shell	90
Altre variabili	92
USO AVANZATO DI VI ED ED	95
Il comando di sostituzione s	95
Il metacarattere	96
Il metacarattere \$	97
Il metacarattere ^	97
Il metacarattere *	98
I metacaratteri []	100
Il metacarattere &	100
Sostituire a capo	101
Modificare linee	101
Shell in editor	102
Ulteriori caratteristiche di vi	103
PREPARAZIONE DI DOCUMENTI	105
 CAPITOLO 3 : I comandi	 113
CAT, PR, MORE	114
CONTENUTO DI DIRECTORY : LS	118

RM	120
CREAZIONE DI DIRECTORY	121
COPIE: CP	122
MV	122
PASSWORD	123
DOCUMENTAZIONE IN LINEA : MAN	125
CHMOD	126
MESSAGGI TRA UTENTI : WRITE, MAIL	128
PARTI DI FILES: HEAD, TAIL	130
SPEZZARE FILES : SPLIT	131
CONTEGGI SU FILES: WC	132
INTERVALLI DI TEMPO: SLEEP, AT	133
PROCESSI E LORO FINE: WAIT, PS, KILL.....	135
ORDINAMENTO DI FILES: SORT, TSORT	138
OCCUPAZIONE DI MEMORIA: DU, DF, QUOT, LCOMPACT	141
COMPARAZIONE FILES: CMP, DIFF	144
RICERCA DI PATTERNS: GREP, LOOK	147
CONDIZIONI: TEST	151
GESTIONE DEL VIDEO TERMINALE: STTY, RESET.....	152
CONTENUTO DI FILES: SEE, STRINGS, FILE	155
RIPETIZIONI DI LINEE: UNIQ	156
REPERIMENTO FILES: FIND	157
PRIORITA': NICE	159
SHELL INTERATTIVA: VSH	160
ALTRI COMANDI	168

CAPITOLO 4 : Shell e cshell 171

LA SHELL	171
Parametri posizionali	171
Strutture di controllo - for -	173
Strutture di controllo - Case -	175
Le variabili di shell	179
Stato di uscita di un comando - test	180
Strutture di controllo - if	181
Strutture di controllo - until e while -	182
Qualcosa di piu'	185
Le redirectioni	185
Parametri posizionali	186
Variabili	187
Ordine di valutazione di un comando	188
Prevenzione dell'espansione	190
Gestione degli errori	191
LA CSHELL	192
Parole	193
Variabili	193
Comandi	194
Correzioni di comandi	197
Alias	197
La sostituzione di nomi di files	201
La sostituzione di comandi	201

Redirezioni	201
Espressioni	202
Strutture di controllo	204
Foreach	204
If	205
while	205
switch	206
Un programma di esempio	207
I files di comandi	207
Ricerca di un nominativo: chenum	207
Inserimento di un nominativo: inserisci	208
Rimozione di nominativi: rimuovi	209
Correzione degli errori	210
 CAPITOLO 5 : Strumenti software	 213
AWK	213
Uso di awk	213
Struttura di un programma	214
Record e campi	214
Scrivere	215
Modelli	217
I modelli BEGIN ed END	218
Espressioni regolari	218
Espressioni relazionali	219
Combinazioni di modelli	220
Intervalli di modelli	220
Le azioni	220
La funzione "length"	221
Variabili, espressioni e assegnamenti	221
Le variabili che indicano campi	223
Concatenazione di stringhe	223
Array	224
Strutture di controllo	224
YACC	226
 CAPITOLO 6 : Il file system	 227
IL FILE SYSTEM	227
Area di bootstrap	227
Il superblocco	228
I-list	229
Blocchi liberi	230
Directory	232
Organizzazione del file system	233
Puntatori di I/O	234
Mount e unmount	237
I FILES SPECIALI	239
Tipi di files speciali	240

Device primari e secondari	243
Protezione dei files speciali	244
 CAPITOLO 7 : Rapide consultazioni	 247
 GLOSSARIO DEI TERMINI	 247
RIASSUNTO DEI COMANDI	257
Shell	257
Manipolazione files e directory	257
accessi	259
Documentazione	260
Terminale	260
Comunicazioni	261
Situazione, processi e memoria	261
Ricerche, filtri	262
Preparazione di documenti	264
Linguaggi	265
Costruzione di compilatori	266
COME E' STATO SCRITTO IL TESTO	267
BIBLIOGRAFIA	272

INTRODUZIONE

Fare oggi un libro in italiano su UNIX (*) e' una necessita': UNIX infatti si sta velocemente affermando su piccoli e medi elaboratori per le sue caratteristiche di flessibilita', potenza operativa e capacita' didattica, cosi' da poter soddisfare ogni tipo di utente, dallo studente al primo approccio con il calcolatore al progettista di sistemi operativi.

Uno dei motivi che ci ha spinto a pubblicare questo volume e' quello di mettere a disposizione esperienze e nozioni normalmente circoscritte all'area degli addetti ai lavori. Lo scopo che ci proponiamo e' quello di agevolare la comprensione del sistema operativo senza dover sempre ricorrere a sintetici (ed a volte ostici) manuali in inglese. Reperire documentazione costituisce un problema non indifferente, mentre noi riteniamo che la cultura informatica non possa appartenere a pochi privilegiati, che troppo spesso costudiscono gelosamente le proprie conoscenze.

La filosofia di UNIX segna un passo in avanti nell'avvicinamento tra il linguaggio umano e il linguaggio della macchina. La logica 'astrusa' della macchina sta facendo posto ad una logica piu' vicina al modo di pensare umano. Per cui, rispetto agli anni passati, si cerca di avvicinare la macchina al linguaggio dell'uomo piuttosto del contrario.

UNIX e' nato nel 1969 dai Bell Laboratories. Gli intenti iniziali del suo creatore, Ken Thompson, erano quelle di creare un sistema operativo che facilitasse le esigenze di chi in varie maniere produce software. Per parecchi anni il suo utilizzo e' stato circoscritto ad ambienti di ricerca ed universitari, che contribuirono non poco a far conoscere ed apprezzare UNIX in ambiti applicativi o gestionali. Dal 1969 UNIX e' passato attraverso molte versioni, ed e' tuttora in fase di ricerca di nuove implementazioni ed aggiunte.

* UNIX e' un marchio registrato della Bell Laboratories.

Le finalita' di questo testo sono di dare, per gradi, una visione metodica e sistematica di UNIX, cominciando con un approccio molto semplice utile per studenti che hanno le prime conoscenze dell'elaboratore, passando attraverso la conoscenza di un uso piu' approfondito del sistema sia dal punto di vista 'utente' che dal punto di vista 'macchina'. L'ultima parte rappresenta un'introduzione all'utilizzo piu' avanzato del sistema. In appendice al testo ci sono alcuni riassunti sui comandi del sistema operativo, un utile glossario dei termini e la descrizione di come e' stato scritto il libro attraverso l'utilizzo di un word processors. Certamente non si puo' diventare sistemisti con un libro, ma esso puo' diventare un valido aiuto nella comprensione e nell'uso di UNIX. E' essenziale per la piena comprensione del tutto, che le indicazioni date sui comandi siano verificate nel concreto, davanti al terminale. Lo UNIX di cui si tratta in questo libro non fa parte di una specifica versione: esso e' orientato piu' che altro alle produzioni dell'universita' californiana di Berkeley.

E' doveroso adesso spendere due parole su che cos'e' un sistema operativo. Ogni elaboratore ha un proprio sistema operativo, che ha la funzione di interfaccia tra l'utente e la macchina. Un sistema operativo e' percio' un grosso programma che consente all'utente di dare direttive ed istruzioni all'elaboratore per la realizzazione di un proprio programma o progetto.

UNIX si presenta come un sistema operativo multiutente (anche se ci sono in progetto versioni monoutente) per cui si presenta il problema della protezione dei propri files da altri utenti. Questo problema nel passato e' stato risolto in varie maniere :

- 1) Tutti i files erano accessibili a tutti
- 2) Ognuno poteva accedere solo ai propri files

UNIX lo ha risolto cosi': ognuno sceglie chi e come puo' accedere ai propri files. Ogni utente fa parte di un gruppo di utenti, tutti i gruppi formano gli utenti complessivi del sistema. Ogni singolo file ha due proprieta': la prima il proprietario stesso e l'altra il gruppo a cui appartiene il proprietario. Il proprietario decide se i propri files possono essere leggibili (solo visione o stampa), scrivibili (modificabili o cancellabili) o eseguibili da chi vuole lui. Questo perche' ogni file ha in se' delle informazioni, modificabili dal proprietario, che regolano l'accesso. Per esempio, un utente potrebbe scegliere di rendere leggibili ed eseguibili i propri files dal gruppo a cui appartiene e

solo leggibili a tutti gli altri utenti.

I comandi di UNIX si presentano "compatti", la sintassi dei comandi segue degli standard. L'istruzione tipo di UNIX e' la seguente:

<comando> <opzioni> <argomenti>

dove

<comando> e' l'istruzione data (che puo' essere il richiamo di un compilatore , dell'editor di sistema, la visualizzazione o la stampa del contenuto di un file o centinaia di altre cose),

<opzioni> sono 'variazioni' al comando stesso

<argomenti> sono i files su cui deve intervenire il comando

Con questa logica, e' possibile talvolta ricostruirsi alcuni comandi senza conoscere la sintassi esatta del comando stesso.

Una importantissima caratteristica su cui si basa la filosofia del sistema e' il concetto di file. Files, terminali, memoria di massa, drives, praticamente qualsiasi entita' e' vista da UNIX come file. Questo fa si che l'utente abbia piena cognizione delle risorse del sistema e della loro dislocazione (della struttura cioe' del file-system) oltre a rendere omogenea la struttura del sistema. UNIX visto dall'utente e' completamente nascosto in quelle che sono le sue attivita' 'interne', la macchina non e' cioe' visibile dall'esterno. Per cui, su macchine diverse, UNIX si presenta alla stessa maniera. Un'altra caratteristica particolare e' la gestione dei processi di UNIX, ma di questo se ne parlera' piu' avanti, per ora basti sapere che un processo e' l'esecuzione di un programma.

La sovrastruttura del sistema operativo che interpreta i comandi dati dall'utente, e' chiamata shell. La shell e' un programma che interpreta ed espande i comandi dati dall'utente sotto forma 'compatta'. La shell si fa carico di :

- * richiamare e caricare in memoria il programma richiesto dall'utente

- * personalizzare il sistema operativo a seconda delle

richieste dell'utente

- * espandere i metacaratteri (caratteri che per la shell hanno un significato speciale)

- * offrire un job control language, strutture di controllo tipo if ... then ... else , for .. in ... per poter gestire l'andamento dei processi.

Per quest'ultima caratteristica, la shell puo' essere usata come vero e proprio linguaggio di programmazione con la caratteristica che le proprie istruzioni sono comandi del sistema operativo.

Ringraziamo l'Istituto di Cibernetica di Milano. Ringraziamo inoltre tutti quei docenti e studenti che con i loro consigli e le loro critiche hanno contribuito a definire gli argomenti trattati.

CAPITOLO 1

Conoscere UNIX

1.1. IL FILE SYSTEM

Questo capitolo tratta della struttura esterna del file system, rimandando la discussione su quella interna (la parte non visibile dall'utente) all'ultima parte del libro. Il file system costituisce l'interfaccia tra l'utente e i dispositivi di I/O (input-output). Qualsiasi entita' di UNIX e' vista come file, per cui tutto l'I/O e' indipendente dal dispositivo fisico in cui avviene ed e' trattato in maniera identica. I files del file system sono di tre differenti tipi : file normali, directory e file speciali.

1.1.1. File normali

Un file normale e' strutturato come una sequenza di byte. UNIX organizza i files in blocchi di 512 byte. Un file di 513 caratteri (un carattere viene codificato in un byte) occuperà così 2 blocchi di memoria. Questo però e' del tutto invisibile all'utente.

root. Questo perché in UNIX esiste il concetto di current directory (cd), cioè di directory in cui si sta lavorando. La cd è sempre definibile dall'utente. Per cui, se la mia directory corrente è ivo, per riferirmi al file "mio", basta il nome finale "mio". Invece se la mia cd è usr, mio è riferito con ivo/mio. Ogni directory ha sempre due files al proprio interno, creati automaticamente dal sistema al momento della immissione di una nuova directory: il file "." che indica la directory stessa ed il file ".." che indica la directory padre, la directory superiore in cui è contenuta la directory stessa. Nella figura 1, usr è la directory padre della directory di nome ivo. Se la mia cd è ivo, per riferirmi al file giulia, posso dare

../giulia

piuttosto che /usr/giulia poiché "." della directory ivo è appunto "usr". Analogamente, la root è la directory padre di usr, dev e bin.

1.1.3. File speciali

Sotto il termine di file speciali vengono indicati tutti i dispositivi fisici e logici di I/O. Unità a disco, memoria centrale, cassette magnetiche, terminali video e stampanti sono trattati alla stessa maniera di un qualsiasi file UNIX, per cui vengono molto semplificate tutte le operazioni di I/O sulle periferiche. I files speciali risiedono sempre nella directory "/dev", quindi per scrivere su una periferica il meccanismo è lo stesso della scrittura su un file normale.

1.2. IL MECCANISMO DELLE PROTEZIONI

Ad ogni file UNIX è associato un proprietario (utente del sistema), un gruppo di appartenenza (il gruppo di cui fa parte il proprietario) e dieci bit di protezione che regolano l'accesso al file. Il gruppo è un insieme di utenti che presumibilmente stanno eseguendo uno stesso progetto o che in qualche modo hanno caratteristiche comuni.

Il carattere in prima posizione indica la natura del file, cioè se è una directory, un file normale o un file speciale. I restanti nove regolano l'accesso in lettura, scrittura ed esecuzione per il proprietario (i primi tre), gli appartenenti al gruppo (i secondi tre) e tutti gli altri utenti (gli ultimi tre). Vediamo cosa vogliono dire i permessi di lettura, scrittura ed esecuzione nei casi di file normali e directory.

	file normali -----	directory -----
lettura	visualizzare il file, stamparlo	elencare i file contenuti in essa
scrittura	modificare o rimuovere il file	creare nuovi file, distruggerne altri
esecuzione	eseguire il file	accedere ai file contenuti senza poterli listare

Per gli operatori o i membri dello staff di gestione del sistema, non esistono limitazioni di accesso ai files.

Questo è un esempio di lista del contenuto di una directory

```
marco sist  drwxrwxr-x 2    128 Nov  9 18:53 .
marco sist  drwxrwxr-x 5    672 Nov  5 14:28 ..
marco sist  -rw-rw---- 1 14114 Nov 10 14:53 prog
marco sist  -rwxrwxr-x 3   5963 Nov  8 17:49 comp
marco sist  drwx----- 2     64 Nov  6 12:23 dir
marco sist  -rw----- 1    824 Nov  8 18:58 schema
```

Il primo campo (la separazione tra i campi è data da uno o più caratteri bianchi) indica il proprietario del file, il secondo il gruppo di appartenenza, il terzo i bit di protezione, il quarto il numero di link (spiegazione più avanti), il quinto il numero di caratteri contenuti nel file, il sesto la data di ultimo aggiornamento e il settimo il nome del file. I file "." e ".." sono due directory che si riferiscono rispettivamente alla directory attuale e alla directory padre.

Se uno dei dieci caratteri del terzo campo è un trattino, vuol dire che quel file non ha il permesso corrispondente alla posizione in cui si trova il trattino stesso. Ad esempio, il trattino in nona posizione presente nel file ".", segnala che non esiste il permesso di scrittura per gli utenti che non appartengono al gruppo. Entrambe le directory "." e ".." hanno i permessi di lettura (r), scrittura (w) ed esecuzione (x) sia per il

proprietario che gli appartenenti al gruppo, inoltre hanno il permesso in lettura ed esecuzione per tutti gli altri utenti. Invece la directory "dir" e' ad uso personale di marco, poiche' ha i tre permessi solo per il proprietario. Questo vuol dire che un utente del gruppo sistemi puo' creare o rimuovere files nella directory in figura, ma non ha accesso alla directory "dir". Alla stessa maniera, il file "schema" e' accessibile solo a marco, mentre il file "comp" e' eseguibile e leggibile da tutti gli utenti e scrivibile dal proprietario e dal gruppo.

1.3. DA LOGIN A LOGOUT

Entriamo a questo punto nel concreto di UNIX, sperando che voi possiate disporre di un terminale interattivo su cui lavorare e ovviamente di un codice per poter usufruire le risorse della macchina. E' possibile, a causa delle diverse implementazioni di UNIX su vari elaboratori, che alcune parti descritte nel testo non corrispondano esattamente.

Un terminale in "attesa" di un utente presenta normalmente la scritta

```
login:
```

A questo punto dovreste digitare la vostra user-id, trasmettendo la scritta al terminale con il tasto <return> o <enter> in alcuni terminali. La user-id e' un codice in vostro possesso o condivisibile con altri che permette di avere accesso alle risorse della macchina. Se la vostra user-id richiede una password (una parola d'ordine conosciuta solo dall'utente per la protezione del proprio lavoro), il sistema rispondera' con

```
Password:
```

ora dovreste digitare la password opportuna per ottenere l'accesso al sistema. La password viene digitata "al buio", cioe' su terminale non si vedono i caratteri digitati (questo per ulteriore protezione). A questo punto se il sistema da'

```
Login incorrect
login:
```

vuol dire che avete commesso un errore nella scrittura della login o della password e dovete ripetere la procedura da capo. Se tutto e' andato bene, vi si presentera' la data in

cui vi siete collegati l'ultima volta (se non e' la prima volta che viene usata quella login) e probabilmente il messaggio del giorno, un avviso scritto dagli operatori del sistema in cui generalmente si danno indicazioni sul nuovo software disponibile, sul funzionamento del sistema o varie altre informazioni. Poi vi si presentera' il "prompt", un carattere o una serie di caratteri che segnalano che il sistema e' pronto ad accettare le vostre richieste. Il prompt che presumibilmente vedrete e' rappresentato dal carattere "\$" o dal carattere "%".

Ora potete dare le vostre istruzioni, la shell e' pronta ad interpretare i vostri comandi fino a che non decidete di scollegarvi dal sistema dando l'istruzione

login

o il carattere <ctrl>D (control d) che si ottiene premendo il tasto control e contemporaneamente il tasto d. A questo punto un altro utente o ancora voi stessi potete nuovamente collegarvi.

1.4. PRIME ISTRUZIONI

Ora vi trovate nella vostra home directory, la directory su cui vi collegate, di vostra proprieta' e in cui potete lavorare liberamente. La shell e' in attesa di un vostro comando, legge la linea/e ed interpreta ed esegue il comando. Un comando e' un programma eseguibile. La shell come prima cosa guarda se nella current directory esiste un programma con il nome del comando dato, se non e' stato trovato, va a cercare il nome nelle directory di sistema che contengono i programmi UNIX.

Il nome di un comando e' sempre la prima parola che compare in una linea di input, poi possono essere presenti opzioni ed argomenti. Dopo la terminazione di un comando, la shell ripropone sempre il prompt. Ora, provate a digitare l'istruzione

who

trasmettendola con <return>. Questa istruzione riporta chi e' collegato al sistema, su quale terminale e la data di inizio collegamento. Per cui who riporta qualcosa del genere

```
opr      console Nov 10 17:11
marco    tty3     Nov 10 17:51
giulia   tty4     Nov 10 17:27
ospiti   tty6     Nov 10 15:12
```

1.4.1. Errori nello scrivere

Se viene digitato

vho

al posto di who, il sistema risponde con la diagnostica d'errore

vho: not found

che intende dire "non ho trovato il comando vho". Il tasto <back-space> del vostro terminale puo' essere usato per tornare indietro sulla stessa riga e correggere eventuali errori. Il carattere @ annulla tutta la riga digitata. UNIX ha un sistema type-ahead, cio' vuol dire che e' possibile dare input durante l'output di un comando ed i caratteri digitati appariranno contemporaneamente insieme all'output. In questo modo e' possibile dare un'istruzione senza aspettare che sia finita la precedente.

Per interrompere l'esecuzione di un programma, bisogna premere il tasto <delete> o o <interrupt>.

1.4.2. Date

Provate ora a dare l'istruzione

date

e vedrete qualcosa come

Thu Nov 11 14:58:33 GMT+1:00 1982

E' possibile dare piu' istruzioni su una riga, separando un'istruzione dall'altra con un punto e virgola come

who;date

1.4.3. Opzioni, argomenti

Cominciamo ora ad introdurre le opzioni e gli argomenti nella stesura di un comando. Esiste un'istruzione chiamata "man", che riporta il manuale del comando specificato. Ora, se volete leggere il manuale del comando who, digitate

```
man who
```

Non disperate se dovrete attendere l'output del comando, poiche' generalmente i manuali in linea, per ragioni di spazio, sono in forma compattata per cui l'istruzione man ha al suo interno un altro programma che scompatta i file (operazione piuttosto lenta). La parola who nell'istruzione precedente e' l'argomento del comando man, ed e' sostituibile da qualsiasi altra parola che corrisponda al nome di un'istruzione. Se volete che l'output di "man who" non "sfugga via" ma si fermi a seconda delle vostre richieste, bisogna dare

```
man -c who
```

in questo modo, durante l'output, dando un <return> si prosegue di una riga nella visualizzazione del manuale e dando uno spazio bianco di una pagina. La parola "-c" rappresenta un'opzione del comando man. Esistono altre opzioni possibili, dare

```
man -c man
```

per saperne di piu sul comando.

1.4.4. Creazione files

Per poter scrivere un programma o un documento o qualsiasi altro testo, dobbiamo poter creare un file di caratteri nella maniera piu' comoda possibile. Per soddisfare questo scopo ci sono dei programmi chiamati editor. Ci sono vari editor sotto UNIX, quello di cui ci occuperemo noi e' l'editor chiamato vi (che sta per visual


```

a      ( istruzione per appendere un testo )
questo e' un testo di prova
per l'utilizzo di ed
-      ( il punto e' un segnale di fine inserimento )
wq     ( aggiorna il tasto su disco ed esce da ed )

```

ed prima di uscire dara'

49

il numero di caratteri scritti. E' stato ora creato nella current directory un file chiamato "prova" di 2 linee e 49 caratteri.

1.5. ISTRUZIONI PER TRATTARE I FILES

Eseguendo l'istruzione di lista

```
ls
```

avremo

```
prova
```

il file appena stato creato. Per avere maggiori informazioni sui file, dare il comando

```
ls -l
```

e si avra'

```
-rw----- 1      49 Nov 11 16:25 prova
```

Se e' stato creato un altro file chiamato prog, con ls -l avremo

```

-rw----- 1    1222 Nov 11 17:44 prog
-rw----- 1      49 Nov 11 16:25 prova

```

1.5.1. Visualizzazione file: cat, pr, more

L'istruzione "cat" e' la piu' semplice dei programmi di stampa files. Essa trasferisce su terminale i caratteri contenuti nei file indicati dagli argomenti.

```
cat prova
```

visualizza il file prova

```
cat prova prog
```

visualizza prima il file prova poi il file prog, concatenandoli assieme. Usare il carattere <ctrl>S se si vuole fermare momentaneamente l'output ed il carattere <ctrl>Q per riprenderlo.

"pr" a differenza di cat, produce un output formattato. "pr" produce ad ogni pagina di output un'intestazione con data, numerazione delle pagine ed il nome del file. Provate ad eseguire

```
pr prova prog
```

Pr inoltre puo' produrre un output su piu' colonne,

```
pr -3 prog
```

stampa il file prog su 3 colonne. L'istruzione more e' la piu' elaborata delle tre, la piu' usata per video terminali. Essa permette di fermare l'output a piacere, di conoscere la percentuale dei caratteri visti sulla totalita' del file, di passare tra un file e un altro in avanti e in indietro, di cercare una certa parola e di eseguire istruzioni del sistema operativo all'interno dell'esecuzione del programma. Provate ora

```
more prova prog
```

Per scorrere in avanti di una riga usare <return> e uno spazio per avere la pagina successiva. Digitare h (sta per help) per avere la lista dei comandi possibili da dare a more.

1.5.2. Altre istruzioni sui files

Si puo' spostare un file da una directory ad un'altra o cambiargli il nome con l'istruzione mv

```
mv prova inizio
```


cambia il nome del file prova in "inizio" e da questo momento bisogna riferirsi ad "inizio". Il nome di un file puo' essere al massimo di 14 caratteri, non deve contenere il carattere / (che e' usato per separare le directory nei path name) e possono inoltre sorgere problemi usando caratteri tipo

* \$ & [?

od altri, poiche' hanno significato speciale per la shell. Per copiare un file, per avere cioe' un duplicato di esso, esiste l'istruzione cp

cp inizio copia

crea un file chiamato "copia" contenente al proprio interno gli stessi caratteri di "inizio". Attenzione: se fosse esistito un precedente file chiamato copia, il suo contenuto originale sarebbe andato perso. A questo punto la lista (ottenibile con ls -l) e'

```
-rw----- 1      49 Nov 11 18:57 copia
-rw----- 1      49 Nov 11 16:25 inizio
-rw----- 1    1222 Nov 11 17:44 prog
```

Infine, quando un file non ci serve piu', possiamo rimuoverlo con l'istruzione rm

rm copia

rimuove il file copia.

1.6. GLI EDITOR VI ED ED

Vi e' un visual editor, cioe' e' possibile muoversi liberamente lungo la pagina e lungo il testo. Ed e' un editor "a linee", per qualsiasi modifica e' necessario riferirsi alla linea indicata. In vi il testo, o una parte di esso, e' sempre presente su video, in 'ed cio' succede solo se esplicitamente richiesto e non e' possibile andarlo a modificare direttamente. Quando siamo in editor di un file, l'editor non modifica direttamente il file su cui si sta lavorando. L'editor fa una copia del file in un'altra parte del sistema ed il contenuto originale non viene modificato fino a che non si da' l'apposito comando di aggiornamento. Per la primissima spiegazione degli editor, faro' un esempio di semplice programma pascal con rimozione degli errori di compilazione. Questo per comprendere i

comandi di modifica del testo. L'approfondimento del compilatore pascal viene rimandato piu' oltre. Il file name di un programma pascal deve sempre finire col suffisso ".p". Il programma in questione somma semplicemente dei numeri. Chiameremo dunque il file somma.p.

1.6.1. Vi

Per creare il file daremo

```
vi somma.p
```

diamo il comando di inserimenti "i" ed inseriamo il testo in Fig.5. Proviamo ora a compilare il programma con l'istruzione

```
pi somma.p
```

("pi" sta per pascal interpreter) ed otteniamo (vedi Fig.6). Nella stesura del programma abbiamo commesso 3 errori :

- 1) la prima linea va modificata in
 program somma(output);
- 2) nella quinta e settima linea, la variabile
 k non e' stata dichiarata per cui dobbiamo
 modificare la seconda linea in
 var j, k : integer;
- 3) alla fine della settima linea va inserito un ";"

```

program somma;
  var j : integer;
begin
  writeln('Questo programma effettua alcune somme ');
  for k :=1 to 10 do
    begin
      j:=j+k
      writeln('La somma e'', j);
    end
  end.

```

Fig. 5 Testo creato con l'editor vi.

```

Wed Nov 17 14:54 1982  somma.p:
   1      program somma;
E  -----^----- Expected '('
   5      for k :=1 to 10 do
E  -----^----- Undefined variable
   8      writeln('La somma e'', j);
e  -----^----- Inserted ';'
In program (null):
  E - k undefined on lines 5 7

```

Fig. 6 Errori in compilazione.

4) Inoltre voglio modificare in quarta linea la parola alcune con la parola dieci.
 Rientriamo in vi
 vi somma.p

Possiamo muoverci lungo il testo

```

      in basso con <return> o con j
      in alto con k
      a destra con <space> o con l
      a sinistra con <back-space> o con h

```

Quando diamo un comando non esistente comparirà per un attimo a fine schermo una riga segmentata.

1a modifica

col cursore siamo all'inizio del testo, col carattere \$ il cursore si posiziona all'ultimo carattere della riga, a questo punto con i possiamo inserire

(output)

e dare poi <esc> per terminare l'inserimento. Generalmente ogni comando di modifica termina con <esc>.

2a modifica

ci posizioniamo sul carattere j della seconda riga, diamo il comando a (inserisce a destra del cursore mentre i inserisce a sinistra) e digitiamo

,k<esc>

3a modifica

andiamo alla settima linea, diamo il comando

A

che appende alla fine della riga, e digitiamo

;<esc>

4a modifica

ci posizioniamo sul primo carattere della parola alcune, diamo il comando

cw (sta per change word)

ora comparira' il carattere dollaro alla fine della parola per indicare da dove a dove sono i limiti della parola. Diamo la nuova parola

dieci<esc>

e la sostituzione e' avvenuta. La sostituzione di parola puo' essere arbitrariamente piu' lunga o piu' corta della vecchia.

Ora possiamo compilare ed eseguire il nostro testo che appare cosi':

```

program somma(output);
  var j, k : integer;
begin
  writeln('Questo programma effettua dieci somme ');
  for k :=1 to 10 do
    begin
      j:=j+k;
      writeln('La somma e'', j);
    end
  end.

```

Ora presenteremo tutti i maggiori comandi di vi in modalita' <esc>.

1.6.1.1. Muoversi lungo il testo

La maggior parte di questi comandi accettano prima di esso un numero come fattore di ripetizione. Per esempio dando

6k

il cursore si sposta 6 linee in alto,

8w

si sposta in avanti di otto parole.

Comando	Funzione
<ctrl>D	il cursore si sposta 12 linee in basso
<ctrl>U	il cursore si sposta 12 linee in alto
j	scende di una riga sulla stessa colonna
k	sale di una riga sulla stessa colonna
l	cursore a destra
h	cursore a sinistra
<ctrl>F	cursore una pagina avanti
<ctrl>B	cursore una pagina indietro
G	da solo sposta il cursore alla fine del testo, con un argomento numerico si sposta alla linea indicata. Ad esempio, 32G va alla trentaduesima linea.
I	da solo, va al primo carattere della linea. 20I va al ventesimo carattere.
<ctrl>G	riporta in penultima riga il numero della linea in cui ci si trova, il numero di linee totali del file e la percentuale della linea corrente rispetto a tutto il file.
H	va all'inizio dello schermo, 6H va alla sesta linea
M	va nel mezzo dello schermo
L	va all'ultima linea dello schermo, 5L alla quint'ultima
w	va all'inizio della prossima parola fermandosi nelle punteggiature.
b	va all'inizio della parola precedente fermandosi nelle punteggiature.

e	va alla fine della parola su cui e' posizionato il cursore
W	come w, non si ferma nelle punteggiature
B	come b, non si ferma nelle punteggiature
^	muove il cursore al primo carattere non bianco
O	cursore ad inizio linea
\$	cursore a fine linea

1.6.1.2. Ricerche di parole

Dando il carattere "~/", il cursore si sposta nella penultima linea del terminale ed e' pronto ad accettare una parola che sara' cercata lungo il testo. Se la parola specificata e' stata trovata, il cursore si posizionera' all'inizio di essa, altrimenti ci sara' un messaggio del tipo "Pattern not found". Ad esempio

/begin

seguito da un <return>, cerca la parola "begin" dalla posizione del cursore in avanti, ricominciando dall'inizio quando finisce il file.

?begin

cerca la parola "begin" dalla posizione del cursore all'indietro, ricominciando eventualmente dalla fine del file. Ora,

n

senza <return>, mi trova la prossima ricorrenza del pattern nel testo. Invece

N

senza <return>, mi trova la prossima ricorrenza del pattern nel testo, invertendo pero' la direzione di ricerca. Con

/begin/6

Il cursore si posiziona 6 linee dopo la ricorrenza di begin.

I pattern (le parole da cercare) possono contenere metacaratteri (caratteri che assumono significato particolare) :

^ vuol dire "all'inizio della riga"
 \$ vuol dire "alla fine della riga"
 . vuol dire "qualsiasi carattere"
 [] racchiude dei caratteri

Esempi :

- 1) /^al
 mi trova il pattern "al" solo se si trova all'inizio di una riga.
- 2) ?:\$
 Mi trova (scendendo all'indietro) il pattern ":" solo se si trova a fine riga.
- 3) /x.y
 Mi trova qualsiasi ricorrenza di una x seguita da qualsiasi carattere e poi una y, per cui patterns come x+y, x*y, x-y, ...
- 4) /[0-9]
 Mi trova qualsiasi cifra.
- 5) I metacaratteri possono essere combinati fra loro:
 /^[0-9]a
 mi trova le cifre seguite da una "a" in inizio riga.

Se voglio riferirmi specificatamente a qualche metacarattere, devo farlo precedere da \. Se voglio cercare il pattern

2^12

devo dare

/2\^12

E' possibile fare ricerche limitate alla linea corrente e ad un solo carattere con

f<carattere>

Il comando

f1

mi trova la prima 1 sulla linea corrente

F

funziona come f, con la differenza che cerca prima del cursore.

1.6.1.3. Inserimenti

Comando	Funzione
i	inserisce il testo a sinistra del cursore
a	inserisce a destra del cursore (tipico da usarsi a fine linea)
A	inserisce alla fine della linea
<ctrl>U	torna indietro di una parola mentre si sta inserendo
@	dato in inserimento, torna indietro di cio' che e' stato inserito
o	apre in inserimento una linea sottostante
O	" " " " " sopra

1.6.1.4. Correzioni

Quasi tutti questi comandi permettono un argomento numerico. Nel primo comando, se digitiamo 10x, verranno cancellati 10 caratteri.

Comando	Funzione
x	cancella un carattere
r<carattere>	rimpiazza il carattere originale con un'altro
R	rimpiazza tutti i caratteri dati fino a <esc>
s	rimpiazza un carattere con una stringa (stringa e' una sequenza di caratteri)

5s	rimpiazza una stringa di 5 caratteri con un'altra stringa
dw	cancella una parola davanti al cursore
db	" " " dietro al cursore
dd	cancella una linea
D	cancella sulla linea corrente tutto cio' che si trova dopo il cursore
cw	permette di cambiare il contenuto di una singola parola
cf<carattere>	cambia sulla riga corrente fino a <carattere> incluso
ct<carattere>	come cf, carattere escluso
cc	permette di cambiare una linea
dL	cancella tutte le linee dello schermo sotto il cursore
dG	cancella dalla riga in cui si trova il cursore fino alla fine del file
df<carattere>	cancella sulla linea corrente dal cursore fino al carattere compreso
dt<carattere>	come df, carattere escluso

1.6.1.5. Scrivere caratteri di controllo

Quando in inserimento vogliamo digitare il carattere di tabulazione <ctrl>I o il carattere <esc> (rappresentato da ^C) o altri caratteri normalmente non scrivibili, dobbiamo far precedere il carattere da scrivere dal carattere <ctrl>V.

1.6.1.6. Errori

Se vogliamo riportare la situazione precedente all'ultimo comando dato (per esempio se abbiamo cancellato una linea con dd e ci siamo resi conto che non era da togliere), diamo il comando

u

Per riportare invece la linea corrente nella situazione precedente ad eventuali modifiche, se non ci siamo spostati dalla stessa, diamo il comando

U

Se invece abbiamo disastroato completamente il testo e lo vorremmo riportare alla situazione precedente che entrassimo in vi, con

:q!

esce da vi senza aggiornare le modifiche.

1.6.1.7. Mettere piu' linee in una

Il comando

J

mette la linea sottostante in coda a quella corrente,

3J

fa lo stesso lavoro con 3 linee.

1.6.1.8. Duplicazione di linee, il comando p

Il comando

Y

da solo, carica in un buffer generico la linea corrente,

12Y

carica nel buffer 12 linee dal cursore in poi. Poi possiamo spostarci a piacere nel testo, e digitando

p

vengono ricopiate le linee presenti nel buffer. Ogni p successiva ricopiera' altre volte le linee. Inoltre l'ultima cancellazione di testo finisce in questo buffer generico e con p e' possibile recuperarlo. La p puo' essere usata per spostare linee di testo da una parte all'altra. Per esempio, possiamo dare

10dd

per cancellare 10 linee di testo, posizionarsi poi sul punto voluto e con

p

riportarle intatte.

1.6.1.9. Incolonnamento di righe

I comandi

>>

e

<<

spostano rispettivamente la riga corrente del valore dello

shiftwidth (normalmente 8 spazi) a destra o a sinistra. I comandi

>L

e

<L

spostano a destra o a sinistra tutte le linee dalla posizione del cursore fino alla fine dello schermo.

>G

e

<G

Spostano a destra o a sinistra dalla posizione del cursore fino alla fine del file.

1.6.1.10. Manipolazione di files

Tutti questi comandi devono essere terminati da <esc> o da <return>.

Comando	Funzione
:w	aggiorna il testo con le modifiche
:w<nome>	scrive il testo sul file <nome>
:x,yw<nome>	scrive le linee dalla x alla y sul file <nome>
:wq	aggiorna il testo ed esce dall'editor
ZZ	come :wq
:q	esce dal testo se non sono state fatte modifiche dall'ultimo aggiornamento
:q!	esce senza aggiornare
:e <nome>	entra in editor sul file <nome>
:r <nome>	inserisce nel testo il file <nome> nella riga sotto il cursore

1.6.1.11. Marking

Quando siamo in editor con un testo molto lungo, puo' tornarci utile contrassegnare una certa linea del testo con una "bandierina" che ci servira' per poterci tornare velocemente.

m<carattere>

fa un mark sulla linea corrente. Supponiamo che abbia dato "ma". In qualsiasi parte del testo ora mi trovo, con

'a

ritorno alla linea precedentemente contrassegnata. E' possibile togliere un mark solo riutilizzando lo stesso <carattere> in una linea diversa.

1.6.1.12. Gestione video

Comando	Funzione
<ctrl>L	Ridisegna lo schermo, e' utile quando un programma esterno a vi interferisce con il video, tipo un messaggio pervenuto o l'output di un programma in background.
z<return>	mette la linea corrente all'inizio dello schermo
z.	linea corrente in mezzo allo schermo
z-	alla fine dello schermo

1.6.1.13. Ripetizioni di comandi

Il comando

ripete l'ultimo comando dato, per cui se precedentemente abbiamo inserito una linea la duplica. Se abbiamo cancellato una linea, cancella la successiva. Con un uso adeguato, questo comando può tornare molto utile.

1.6.1.14. Ritrovare linee cancellate, file persi

L'editor salva le ultime 9 cancellazioni di parte del testo in registri numerati da 1 a 9. L'ultima linea cancellata si trova sempre nel registro 1.

"<numero>p

dove in <numero> c'è una cifra da 1 a 9, mi riporta nel file il contenuto del registro indicato. Se quella non è la linea che volevamo, possiamo dare

u

per toglierla (ricordando che u annulla il comando precedente) e poi

.

In questo caso il punto ha il significato di incrementare il numero del registro. A questo proposito provate a dare la sequenza

"1pu.u.u.u.u.u

Se capita che il sistema cade (e' temporaneamente inutilizzabile) mentre siete in editor e vi interessano le modifiche fatte dall'ultimo aggiornamento, generalmente e' possibile ritrovare il file con il comando

```
vi -r <nomefile>
```

1.6.1.15. Comandi per tutto il testo

Introdurremo ora i comandi a livello dei `:`. Dopo aver dato il carattere `:` il cursore si trova nella penultima linea del terminale. Questi comandi sono "ereditati" dall'editor ed, con la differenza che in ed non e' necessario dare il `:`. Provate ora a dare

```
:p
```

verra' cosi' visualizzata la linea corrente. Dando ora

```
:3,6p
```

vengono visualizzate le linee dalla 3 alla 6. Il punto invece indica la linea corrente(la linea su cui si trova il cursore).

```
:. ,+6p
```

stampa la linea corrente e le sei seguenti. Il carattere `$` indica la fine del file, per cui

```
:1,$p
```

visualizza tutto il file. Come

```
:. ,,$p
```

visualizza il file dalla linea corrente in poi. D'ora in poi si omettera' il carattere `:` nei comandi. Dando

```
..,$-3p
```

si avranno le linee dalla linea corrente fino alla terzultima esclusa. Con

```
-5,+5p
```

vedo le 5 linee sopra e sotto al cursore. Visto come funziona il meccanismo di riferirsi ad una parte di testo, possiamo vedere altri comandi oltre quello di print (il `p` alla fine degli indirizzi di testo usato fin'ora). Se alla fine degli indirizzi diamo una `,d`, verranno cancellate le linee indicate. Così'

.,\$d

cancella le linee dall'attuale fino alla fine del file. Vediamo ora il comando di sostituzione s. Questo e' il comando piu' difficile da usare ma e' anche il piu' potente di tutti. Come prima cosa, vediamo la sostituzione di una parola nella linea corrente

s/vecchia/nuova

Sostituisce nella linea in cui si trova il cursore, la prima ricorrenza di "vecchia" con "nuova".

s/vecchia/nuova/g

La g sta per global, e sostituisce qualsiasi ricorrenza di vecchia con nuova nella linea corrente. Ogni comando di sostituzione puo' essere preceduto da una o piu' numeri di linee.

1,\$s/error/errore/g

mi cambia ogni ricorrenza di "error" in "errore" in tutto il testo.

23,55s/error/errore/g

la stessa cosa dalle linee 23 alla 55. Puo' capitare che dopo un comando di sostituzione ci accorgiamo che il risultato non era cio' che desideravamo, in questo caso e' sempre valido il comando

u

per riportarci alla situazione precedente il comando.

1.6.1.16. Parametri

L'editor vi ha un insieme di parametri che possono essere settati dall'utente. Provate a dare l'istruzione

:se all

Questa istruzione vi mostra tutti i possibili parametri che possono essere settati. Dando invece

:se

vengono mostrati i parametri che sono stati attivati. Quando stiamo siamo in editor di un programma, ci puo' essere utile l'autoindentazione. Date il comando

```
:se ai
```

che sta per set autoindent, inserite una nuova linea lasciando alcuni spazi bianchi all'inizio della riga, andate a capo sempre in inserimento e vedrete che la nuova linea comincia dal punto in cui e' cominciata quella precedente. E' cosi' possibile evitare noiosi inserimenti di spazi bianchi all'inizio delle linee. Per tornare nelle colonne a sinistra in inserimento, dare il carattere

```
<ctrl>D
```

Ogni volta che diamo questo carattere come primo carattere non bianco della riga (sempre in inserimento), il cursore si sposta otto spazi piu' a sinistra. Se vogliamo modificare il numero da otto ad un'altro, per esempio tre, diamo

```
:se sw=3
```

dove sw sta per shiftwidth.

Quando vogliamo togliere l'autoindentazione, diamo

```
:se noai
```

1.6.1.17. Cose che capitano in vi

Supponiamo che abbia dato il comando

```
vi .
```

o al posto di . il nome di una directory. Siccome la directory viene gestita direttamente dal sistema, non ci viene permesso di scriverci dentro. Ogni modifica che abbiamo fatto non potra' venire aggiornata e l'unica maniera di uscire da vi e' attraverso il comando

```
:q!
```

Oppure ci puo' capitare di essere in editor di un file in codice (tipo il risultato di una compilazione) e in questo caso vi potrebbe dare il messaggio di errore

```
<nomefile> : line too long
```

cio' vuol dire che la linea logica (una linea finisce con il carattere di nuova linea e non quando la linea e' all'estrema destra dello schermo) e' troppo lunga.

Ci puo' capitare ancora di aver richiamato un file in cui sono presenti caratteri non appartenenti al set ASCII, in questo caso non sono visibili questi ultimi.

In ogni caso molto difficilmente ci interessa modificare file del genere e con il comando ":q!" possiamo sempre uscirne.

Se, dopo aver dato il comando "vi" al sistema, non e' ancora apparsa la pagina ed abbiamo trasmesso il tasto <delete>, puo' capitare che vi ci dica una cosa del genere:

```
At end of file
#
```

o altri tipi di scritte. Questo succede perche' con delete e' stato interrotto o il caricamento del programma vi in memoria centrale oppure il caricamento del testo stesso in un momento delicato. Possiamo uscirne alla solita maniera.

1.6.2. Ed

Per chiamare ed, diamo il comando

```
ed text
```

se gia' esiste il file text, ed risponde con il numero di caratteri contenuti, altrimenti risponde semplicemente con

```
?text
```

ora diamo

```
a<return>
```

e cominciamo ad inserire le nostre linee

```
Ma che piccola storia ignobile
mi tocca raccontare
cosi' solita
e banale
come tante
```

comunicando la fine dell'inserzione con un punto ad inizio linea.

.<return>

Sono state inserite dunque 5 linee di testo. Ora possiamo, tramite opportuni comandi, modificare e/o continuare il testo della famosa canzone. Teniamo presente che ogni qual volta viene commesso un errore nei comandi, l'editor ed lo segnala con

?<messaggio d'errore>

o con un ? semplicemente. Se adesso diamo un altro <return>, ed segnalera' che siamo gia' giunti alla fine del file. Ora diamo

1<return>

ci viene visualizzata la prima linea ed ogni <return> ci visualizza le linee successive fino alla fine del file. Un numero senza altro vuol dire implicitamente "visualizza la linea richiesta". E' possibile dare due estremi di indirizzi di linee. Così'

1,\$

visualizza tutto il testo.

.,+8

visualizza le otto linee dalla linea corrente in poi. I meccanismi di indirizzamento sono quelli spiegati nel paragrafo 6.1.15.

La linea corrente per ed e' l'ultima linea visualizzata o inserita o modificata o infine trovata con una ricerca.

Tenendo presente che tutti i comandi di vi a livello di ":" sono ugualmente efficaci in ed, vediamo i principali comandi di modifica del testo.

1.6.2.1. Cambiamenti

Il comando "c", seguito da <return>, e' il comando di cambiamento. Questi sono alcuni esempi d'uso autoesplicativi.

```

c
queste 3 righe
cambiano cio' che c'era
nella riga corrente
.

+1,$c
queste 4 righe
cambiano le linee
da quella successiva alla linea corrente
fino alla fine del file
.

```

1.6.2.2. Sostituzioni

Il comando "s" e' il comando di sostituzione. Rifacendosi anche a quello che e' stato scritto in proposito sull'editor vi, verranno analizzati alcuni esempi. Non potendo in ed posizionarsi sulla parola o carattere da modificare, a differenza di vi viene molto usato il comando di sostituzione per una singola linea. Ogni comando di sostituzione ha implicitamente il comando di stampa della linea modificata o, nel caso sia piu' di una, l'ultima sostituita.

Istruzione	Significato
s/a	Sostituisce la prima a che incontra nella linea corrente con niente, praticamente e' il corrispondente di x in vi
s/caso/casa	Sostituisce la prima ricorrenza di caso con casa nella linea corrente
5s/caso/casa	Sostituisce la prima ricorrenza di caso con casa nella quinta linea
1,\$s/caso/casa/g	Sostituisce qualsiasi ricorrenza di caso con casa in tutto il testo
s/\$/ fine/	Sostituisce alla fine della linea corrente la parola " fine", cioe' aggiunge la

parola alla fine della riga,
e' l'equivalente del comando
A di vi

s/^/inizio /

Sostituisce all'inizio riga la
parola "inizio ", inserisce
cioe' prima del primo carattere
della linea. E' l'equivalente
di I in vi

1.6.2.3. Ricerche

Dopo ogni ricerca viene visualizzata la linea o le linee cercate. E' possibile combinare le ricerche con comandi di sostituzione, di cancellazione, con il comando g (global) e con il comando v (ricerca delle linee che non contengono il pattern).

Se il comando "g" e' presente come primo carattere di una linea di comandi, significa "in tutto il testo", se e' presente come ultimo carattere, significa "in tutta la linea".

Il significato dei metacaratteri

^ \$ [-]

e' uguale a quello delle ricerche in vi.

Il backslash (\) come al solito serve per togliere il significato speciale ai metacaratteri.

Istruzione	Significato
/err/	Cerca la linea che contiene err dalla linea corrente in poi, segnalando con "?failed" se non compare nel testo
?err?	Come sopra, esegue la ricerca all'indietro
//	Cerca in avanti la prossima ricorrenza del pattern
??	Come sopra all'indietro
/err/i	Entra in inserimento prima della linea che contiene err

<code>/err/s//rer</code>	Sostituisce rer alla linea che contiene err. I due // significano "cio' che e' stato prima specificato.
<code>g/al/</code>	Visualizza tutte le linee che contengono al
<code>/al/d</code>	Cancella la prima linea incontrata che contiene al
<code>g/al/d</code>	Cancella (d sta per delete) le linee che contengono al
<code>v/al/d</code>	Cancella tutte le linee che non contengono al
<code>/[0-9]/</code>	Cerca qualsiasi cifra
<code>g/[ACG]/d</code>	Cancella tutte le linee che contengono A, C o G
<code>/^R/</code>	Cerca una linea che inizi con R
<code>/^unica\$/</code>	Cerca una linea che contenga solo la parola "unica"
<code>/3\\$/</code>	Cerca una linea che contenga 3\$

1.6.2.4. Altri comandi

I comandi di manipolazione files sono praticamente identici a quelli di vi con qualche differenza: il comando che permette di uscire senza aggiornare

- in vi q!
- in ed Q

l'introduzione di un file esterno nel testo :

- in vi con `:r <nomefile>` viene inserito nella linea corrente
 - in ed con `r <nomefile>` viene inserito alla fine del testo
- per inserirlo ad una certa linea, bisogna specificarla:
- `!4r <nomefile>` inserisce il file dopo

la quattordicesima linea

Il comando m (move) serve per spostare linee da una parte all'altra del testo. Esempi :

5,15m20

sposta le linee dalla 5 alla 15 alla linea 20.

1,10m-1

sposta le linee dalla 1 alla decima sopra la linea corrente

m+

sposta la linea corrente (per default se non e' specificato nessun indirizzo) nella linea successiva, praticamente scambia di posizione 2 righe.

Il comando di marking e'

k<carattere>

e per ritornare alla linea contrassegnata

'<carattere>

Per la duplicazione di testi, bisogna dare un comando cosi' strutturato

<linee-da-copiare>t<dopo-la-linea-n.>

Il comando

.,+10t2

copia 10 linee dalla corrente in poi dopo la seconda linea del testo.

Il comando

n

ci da' il numero di linea della linea corrente.

Il comando

f

scrive il nome del file su cui si sta lavorando

L'ultimo comando dato viene annullato con

u

come in vi.

Per metter insieme 2 righe,

j

(in vi era J)

1.7. USO DI DIRECTORY

Ammettiamo che si voglia stendere un grosso documento diviso in vari capitoli, ognuno dei quali corrisponde ad un file. E' piu' comodo allora che tutti questi files risiedano nella stessa directory. Per poter creare una directory, si usa l'istruzione mkdir, per cui

```
mkdir documento
```

mi crea una directory vuota chiamata documento che appare cosi' nella lista:

```
drwx--x--- 2      32 Nov 11 19:25 documento
-rw----- 1      49 Nov 11 16:25 inizio
-rw----- 1    1222 Nov 11 17:44 prog
```

Ammettiamo che la vostra home directory si chiama /usr/utenti/max. Se vogliamo lavorare nella directory appena creato, usiamo l'istruzione

```
cd documento
```

cd sta per current directory e serve per cambiare la directory in cui si lavora. La home directory e' sempre max, ma cambia la current directory e quindi il path name che e' il seguente

```
/usr/utenti/max/documento
```

Esiste a questo proposito l'istruzione

```
pwd
```

che riporta il path name della directory in cui ci troviamo.

Listando la directory, vediamo adesso solo i file "." e "..", il cui significato ci e' gia' chiaro. Vediamo ora

alcuni di esempi d'uso di questi file. Essenzialmente servono per poter comunicare tra directory padre e figlia. Se noto che il file "prog" avrebbe la sua logica collocazione nella nuova directory, posso spostarlo dando

```
mv ../prog .
```

Cioe' : sposta il file chiamato prog, residente nella directory padre ("..") nella directory corrente ("."). Se voglio ora listare la directory padre, faro'

```
ls -l ..
```

ed ho

```
drwx---x--- 2      48 Nov 11 19:37 documento
-rw----- 1      49 Nov 11 16:25 inizio
```

Se voglio visualizzare il file "inizio" stando sempre nella directory documento, daro'

```
more ../inizio
```

Se adesso voglio ritornare nella directory di partenza, posso usare

```
cd ..
```

o piu' semplicemente

```
cd
```

Quest'ultima sposta la current directory nella directory di home, qualsiasi sia la directory in cui ci troviamo.

1.8. METACARATTERI

Supponiamo ora di aver creato un documento suddiviso per capitoli e per paragrafi. La lista (ls) appare cosi' :

```
cap.1.1 cap.1.2 cap.1.3 cap.1.4
cap.1.5 cap.1.6 cap.2.1 cap.2.2
cap.2.3 cap.2.4 cap.3.1 cap.3.2
cap.3.3 cap.4.1 prog
```

Esistono in UNIX delle possibilita' che permettono di semplificare alcune noiose operazioni. A questo scopo esistono dei metacaratteri che, opportunamente interpretati

dalla shell, sintetizzano lunghi elenchi di files. Se vogliamo stampare il documento intero, possiamo dare l'istruzione

```
pr cap.1.1 cap.1.2 cap.1.3 cap.1.4 cap.1.5 cap.1.6 cap.2.1 ...
```

elenicare cioè tutti i files. E' chiaro che ci si stanca facilmente dovendo dare 14 nomi di files, senza contare i probabili errori di battitura. L'istruzione sopra può essere così sintetizzata

```
pr cap*
```

o anche

```
pr c*
```

Il carattere * significa "qualsiasi cosa", in questo caso qualsiasi cosa dopo cap o qualsiasi cosa dopo c. L'asterisco non e' limitato a doversi trovare nell'ultimo posto di un filename. Se vogliamo stampare tutti i secondi paragrafi dei capitoli, faremo

```
pr *2
```

e in questo caso l'asterisco seleziona i files

```
cap.1.2 cap.2.2 cap.3.2
```

* da solo seleziona tutti i files della current directory (attenzione al comando "rm *" poiché ci rimuove tutti i files presenti nella directory). Se voglio riferirmi solo ai capitoli dal due al quattro, per esempio averne la lista, do'

```
ls -l cap.[234]*
```

ed ho

```
-rw----- 1      2341 Nov 12 14:38 cap.2.1
-rw----- 1      8282 Nov 12 14:39 cap.2.2
-rw----- 1      8321 Nov 12 14:39 cap.2.3
-rw----- 1       231 Nov 12 14:39 cap.2.4
-rw----- 1     4674 Nov 12 14:38 cap.3.1
-rw----- 1     6366 Nov 12 14:40 cap.3.2
-rw----- 1     3838 Nov 12 14:40 cap.3.3
-rw----- 1     9838 Nov 12 14:38 cap.4.1
```

Le parentesi quadre indicano di selezionare qualsiasi carattere all'interno delle parentesi. Una serie di lettere o numeri consecutivi può essere abbreviata all'interno delle parentesi con un trattino. L'istruzione precedente può essere espressa nella seguente forma

```
ls -l cap.[2-4]*
```

Il carattere ? significa "qualsiasi carattere". Per esempio,

```
more cap?.1
```

visualizza i primi paragrafi di ogni capitolo, seleziona i files

```
cap.1.1 cap.2.1 cap.3.1 cap.4.1
```

Per togliere il significato speciale ai metacaratteri, bisogna includerli tra apici, come la seguente

```
ls '[
```

1.9. VISIONE DEL FILE-SYSTEM

Ora andiamo a dare un'occhiata "fuori casa nostra", nelle directory di sistema. E' possibile percorrere il lungo e in largo l'albero del file system, per ottenere informazioni sulla struttura del sistema. Abbiamo detto che l'oggetto da cui parte il filesystem e' la root (/). Proviamo allora a listare la directory /

```
ls -l /
```

produce

```
drwxr-xr-x 2      832 Jun 24 18:02 bin
drwxr-xr-x 2      864 Sep  8 00:15 dev
drwxr-xr-x 3      816 Nov 11 13:36 etc
drwxr-xr-x 2      384 Jun 24 18:02 lib
drwxr-x--x 2      384 Oct 25 15:57 lost+found
drwxr-xr-x 3      112 Nov  3 13:43 priv
drwxrwxrwx 2     1648 Nov 12 17:52 tmp
drwxr-xr-x 16     384 Aug  2 12:22 usr
```

Forse nel vostro sistema le cose sono leggermente diverse, questo dipende da come e' stato configurato il file system da parte degli operatori o da quale versione UNIX e' stata montata. Queste comunque sono le directory principali del sistema, da cui ne derivano altre. La parte veramente riservata all'utente e' la directory /usr oppure una sottodirectory di quest'ultima mentre tutto il resto e' riservato al sistema. Vediamo un po' piu' in dettaglio la

funzione di queste directory

- /bin : contiene in forma di codice macchina i comandi UNIX
- /dev : contiene tutti i file speciali
- /etc : e' una directory preposta alla manutenzione, sviluppo e controllo delle attivita' del sistema, utile soprattutto ai membri dello staff che gestisce il sistema stesso.
- /lib: contiene librerie di funzioni per programmi C, pascal, pl1 od altri linguaggi
- /lost+found : quando cade il sistema o manca la corrente elettrica, UNIX salva il contenuto dei file su cui si stava lavorando in questa directory
- /priv : e' la directory dei membri dello staff di sistema
- /tmp : e' la directory su cui vengono mandati tutti i file temporanei prodotti da comandi tipo vi, ed ed altri di cui vedremo
- /usr : e' normalmente la directory piu' grossa per numero di files ed occupazione di memoria e contiene oltre ai file degli utenti, altri comandi, file di amministrazione, manuali (quelli richiamabili con man), librerie e cose di pubblica utilita'.

1.10. COMUNICAZIONI TRA UTENTI

Ci sono diverse maniere, per poter comunicare tra utenti, la piu' usata e' quella di spedire una lettera all'utente desiderato. Cio' si ottiene con l'istruzione

```
mail <nome-utente/i>
```

Per nome-utente si intende la login con cui un utente si collega al sistema. Digitando

```
mail giulia
```

ci comparira'

Subject :

in questo momento bisogna digitare il titolo della lettera. Poi scrivete il vostro testo finendo il tutto con <ctrl>D. Il carattere <ctrl>D e' il segnale di EOF ed e' usato per comunicare al sistema che l'input e' finito. Quando giulia si colleghera' al sistema, dopo il messaggio del giorno gli comparira'

You have mail.

essa non dovra' fare altro che battere

mail

per leggere cio' che c'e' nella cassetta della posta. Ogni volta che un utente riceve posta, gli viene automaticamente aggiornato un file nella home directory chiamato mbox che contiene tutte le lettere ricevute. Un'altra maniera di comunicazione e' quella offerta dal programma write. L'uso e'

write <nome-utente>

segnalando con <ctrl>D la fine del messaggio. In questo modo il messaggio scritto apparira' sul terminale con cui e' collegato l'utente. Se con la stessa login, sono collegati in due o piu' terminali bisogna allora specificare il numero del terminale (l'istruzione who da queste informazioni) nel seguente modo

write to tty<numero>

per esempio

write to tty3

Se non volete ricevere messaggi, basta dare l'istruzione

mesg n

ripristinando la possibilita' di riceverli con

mesg y

1.11. RIDIREZIONI

UNIX ha tre file associati ad ogni terminale :

```
standard input ( per default la tastiera )
standard output( "      "      il video )
standard error ( "      "      "      "      )
```

Lo standard input e' l'input che viene dato ad un comando (ad esempio nel mail, la scritta prima di <ctrl>D)

Lo standard output e' l'output di un comando

Lo standard error sono i messaggi d'errore se un comando non e' stato portato a termine nella corretta maniera a causa di un errore dell'utente nel comando stesso. Ad esempio, se diamo

```
ls -l xxx
```

senza che esista il file xxx, la scritta

```
xxx: not found
```

va nello standard error (sul video terminale).

E' pero' possibile ridirigere l'input, l'output e l'error da e su altri file al posto del terminale. Se ho creato con vi o con ed un file chiamato messaggio e lo voglio spedire a giulia come mail, posso fare cosi'

```
mail giulia < messaggio
```

il minore in sostanza vuol dire: esegui il comando che mi precede prendendo come input il contenuto del file che segue (messaggio) piuttosto che la tastiera. Abbiamo cosi' ridiretto lo standard input dalla tastiera al file messaggio. Analogamente possiamo ridirigere lo standard output su di un qualsiasi file.

```
who > chi
```

scrive l'output di who sul file chi. Se "chi" era un file gia' esistente, e' stato perso il suo contenuto originale e gli e' stato sostituito il nuovo. Per scrivere invece in coda ad un file senza perdere il contenuto originale, si usano i due caratteri ">>"

```
date >> chi
```

mi accoda la data attuale al file chi

```
cat chi
```

a questo punto mi produce

```
opr      console Nov 10 17:11
marco    tty3     Nov 10 17:51
giulia   tty4     Nov 10 17:27
ospiti   tty6     Nov 10 15:12
Fri Nov 12 18:40:57 GMT+1:00 1982
```

Analogamente posso riunire tutti i capitoli del mio documento in un file unico chiamato TUTTO con

```
cat cap* > TUTTO
```

Per ridirigere lo standard error di un comando, si usa 2>.

```
cat testo 2> sbagli
```

mi manda eventuali errori dell'istruzione cat nel file chiamato sbagli. In questo caso in "sbagli" ci andra'

```
testo: not found
```

se non esiste il file testo nella current directory. Posso combinare tra loro < e >. Createvi un file di comandi per l'editor ed chiamato per-ed con il seguente contenuto

```
1,$s/nome1/nome2/g
$
a
Ultima riga del testo
-
w
q
```

Questi comandi forniti ad ed sostituiscono tutte le ricorrenze di nome1 in nome2 ed appende una riga alla fine del testo. Potete ora eseguire questi comandi su un file (per esempio prog) digitando

```
ed prog < per-ed
```

ora pero' se non vogliamo che ci appaia l'output di ed su terminale (numero di caratteri in entrata ed uscita, ecc..) o se ci interessa salvare l'output stesso, possiamo deviarlo su un file chiamato temp Per cui,

```
ed prog < per-ed >temp
```

oppure, se non ci interessa proprio l'output,

```
ed prog < per-ed >/dev/null
```

/dev/null e' un file speciale che ha la caratteristica di essere e rimanere sempre vuoto per cui puo' essere usato per scaricare output non interessante.

1.12. STAMPANTE

Se volete a questo punto trasferire il contenuto di alcuni vostri files su stampante, potete dare l'istruzione

```
lpr <nomefile>
```

questa istruzione mette la vostra stampa in coda alle richieste di stampa effettuate da altri utenti, e quando sara' venuto il vostro turno, vedrete comparire il vostro tabulato, intestato con il nome della vostra login.

1.13. PIPE

Una pipe e' una maniera per connettere lo standard output di un programma con l'input di un altro. L'output di un programma invece di comparire su terminale, viene letto come input dal secondo programma. Esiste in UNIX un comando che conta le linee, le parole e i caratteri di un file, chiamato "wc" (word count); con l'opzione "-w", conta solo il numero di parole.

```
wc -w prog
```

da in output il numero di parole contenute nel file prog. Ora, se voglio sapere quanti files ho nella current directory, posso procedere cosi' : con

```
ls > lista
```

mando la lista ottenuta con ls sul file chiamato lista, con

```
wc -w lista
```

ottengo il numero di parole del file lista (= il numero dei files)


```
rm lista
```

rimuovo il file lista che non mi serve piu'. Tutto questo puo' venir eseguito con l'unica istruzione

```
ls | wc -w
```

cioe' "prendi l'output di ls come input per wc -w". Su terminale mi compare unicamente il numero di parole di ls senza dover creare ne' rimuovere files temporanei. Così come

```
who | wc -l
```

mi da' il numero di utenti collegati. wc -l conta solo il numero di linee di un file. Ora, se voglio mandare il contenuto di tre files formattati con pr su stampante, posso dare

```
pr file1 file2 file3 > tuttitre
lpr tuttitre
rm tuttitre
```

anche in questo caso la pipe ci puo' semplificare le cose: con

```
pr file1 file2 file3 | lpr
```

ottengo lo stesso risultato. Per la piena comprensione, e' bene che questi esempi siano verificati direttamente davanti al terminale. La pipe non e' limitata ad essere una sola, nel seguente esempio

```
who | wc -l | lpr
```

ho 2 pipe ed ottengo così su stampante il numero di utenti collegati. E' una pipeline.

1.14. PROCESSI

Ogni esecuzione di un programma e' chiamato processo. Ogni processo e' identificato da un numero univoco. Ogni processo lanciato incrementa il numero. Ogni comando dato e' un processo diverso.

```
pr file1 file2 file3
```

e' un processo,

```
who | wc -l
```

sono 2 processi poiche' devono essere eseguiti 2 programmi.

```
who | wc -l |pr
```

sono 3 processi. E' possibile mandare in esecuzione un processo in "background" cioe' fare eseguire il comando senza attendere la fine dell'esecuzione potendo cosi' continuare il lavoro. Esempio:

```
ed prog < per-ed &
```

Il carattere "&" alla fine di un'istruzione significa: "comincia ad eseguire il comando e, senza aspettare che sia finito, accetta altri comandi". Chiaramente, se vogliamo che l'output di "ed" non interferisca con il nostro terminale, faremo meglio a dare

```
ed prog < per-ed > out-ed &
```

Ogni volta che si termina un comando con &, il sistema risponde con il numero del processo attivato. Questo numero puo' essere utilizzato per fare finire l'esecuzione del processo prima della sua fine naturale con l'istruzione

```
kill <numero-processo>
```

Questa istruzione e' utile nel caso in cui un processo va in un loop infinito, quando ci rendiamo conto che stiamo facendo dei disastri o per qualsiasi altro motivo non vogliamo che continui l'esecuzione. Se ci dimentichiamo il numero del processo, il comando "ps" ci riferisce quali programmi sono in esecuzione

```
ps
```

produce la Fig.7. Ps mi riporta il numero del processo, il terminale su cui e' stato mandato in esecuzione, il tempo di cpu e il nome del comando. La prima riga si riferisce al

PID	TTY	TIME	CMD
18093	co	0:08	ed
12858	co	0:39	-sh
18159	co	0:02	

Fig. 7 Esempio di output del comando ps.

programma ed mandato in background, la seconda e' la shell, sempre attiva da quando ci si collega a quando si lascia il terminale, la terza si riferisce al processo generato dal comando ps. Ps senza opzioni ci riferisce sui processi dell'utente, per avere informazioni su tutti i processi attivi, dare

```
ps a
```

e si otterra' qualcosa tipo la Fig.8. I processi /etc/cron e /etc/openup sono processi di sistema, -2 indica che il terminale e' in attesa che qualcuno si colleghi. L'istruzione

```
kill 0
```

fa terminare tutti i vostri processi, inclusa la shell.

E' possibile anche fare eseguire dei comandi da un'altra shell, facendo in modo che all'uscita dei comandi vengano ripristinati i parametri precedenti. La sintassi vuole che

PID	TTY	TIME	CMD
15811	7	0:22	-sh
17957	5	0:14	-sh
18326	co	0:02	
18093	co	0:20	
59	co	1:22	/etc/cron
61	co	0:01	/etc/openup
12858	co	0:39	-sh
16990	4	0:15	-csh
18227	3	0:01	
17052	2	0:07	-sh
17577	1	0:09	-sh
18327	5	0:03	vi prova
17065	2	0:02	-sh
17714	2	0:11	vi newton.p
18270	3	0:02	
18338	3	0:00	
18337	7	0:00	
18287	6	0:01	- 2
18331	co	0:09	
18245	3	0:01	
15728	3	0:22	-sh
18267	4	0:04	

Fig. 8 Esempio di output del comando ps a.

suddetti comandi siano racchiusi tra parentesi tonde. Dopo l'esecuzione di

```
( cd dir ; ls -l )
```

non e' stata modificata la nostra current directory.

1.15. . PROTEGGERE I PROPRI FILES

E' utile che ogni utente possenga una propria password, una parola d'ordine che da' accesso ai propri files, conosciuta solo da chi ha diritto di lavorare con quella specifica login. Per mettersi una password, dare l'istruzione

```
passwd
```

Ora, se precedentemente c'era gia' una password, si assume che la vogliate cambiare e dovrete perciò digitare la vecchia. In ogni caso, il sistema vi chiederà due volte di digitare la password facendo in modo che i caratteri non compaiono su video. Sarà confermata solo se e' stata digitata la stessa frase due volte. Per proteggere i vostri files, ovvero sia per cambiare i flag di protezione, si usa l'istruzione

```
chmod
```

Ci sono due maniere di usare questa istruzione : una e' la seguente, dove i caratteri tra quadre indicano le opzioni possibili.

```
chmod [ugoa][+--=][rwxstugo] file ...
```

L'altra e'

```
chmod <numero> file ...
```

Vediamo qualche esempio, rimandando indietro sul significato dei flag. questa e' la lista della directory

```
drwx--x--- 3      160 Nov 16 17:11 .
drwxr-xr-x 5      672 Nov 16 17:59 ..
-rwxrwxrwx 1      855 Nov 12 22:23 prog
```

1a maniera:

a) rendo leggibile ed eseguibile a tutti la directory con

```
chmod go+rx .
```

il punto indica la directory, go sta per gruppo ed altri utenti, (u per il proprietario) il + indica di aggiungergli i flag che vengono dopo. Se ora do

```
ls -l .
```

avro'

```
drwxr-xr-x 3      160 Nov 16 17:11 .
```

b) ora tolgo a prog l'eseguibilita' al proprietario, al gruppo e a tutti gli altri con

```
chmod ugo-x prog
```

In questo caso i - indicano di togliere i flag che seguono. Prog viene cosi' modificato in

```
-rw-rw-rw- 1      855 Nov 12 22:23 prog
```

2a maniera:

In questa maniera si danno tre cifre ottali (una per il proprietario, una per il gruppo ed una per tutti gli altri) che vengono costruite sommando 1 per l'eseguibilita', 2 per la scrittura e 4 per la lettura. Per cui rwx corrisponde a $4+2+1=7$, r-x corrisponde a $4+1=5$ L'esempio a) corrisponde a

```
chmod 755 .
```

mentre l'esempio b) e'

```
chmod 666 prog
```

1.16. IL COMPILATORE PASCAL PI

La prima versione del compilatore pi, detto anche Berkeley Pascal, fu scritta da Ken Thompson nel 1976. Berkeley Pascal ha una caratteristica molto evidente : l'uso didattico che se ne puo' fare, non trascurando tuttavia l'estensione e la potenza di calcolo. Le segnalazioni di errori sia in compilazione che in esecuzione sono molto varie e per esteso, dando cosi' la possibilita' di avere

maggiori facilitazioni nella diagnostica e nella risoluzione di errori. Il risultato della compilazione e' un codice che viene interpretato al momento dell'esecuzione dall'interprete "px". Dovendo interpretare un codice, l'esecuzione sara' piu' lenta di quella di un codice in formato assoluto (in linguaggio macchina), ma la compilazione sara' piu' veloce. L'occupazione di memoria del codice prodotto e' piuttosto limitata. Altra utile caratteristica, e' la quantita' di software disponibile legata al compilatore "pi": profilazioni di esecuzione, indentazione automatica del testo, sottolineatura di parole chiave, cross-reference. Vediamo ora il significato e l'uso di questi programmi.

1.16.1. Primo programma

Un programma e' un testo creato normalmente con un'editor di sistema. Il compilatore pascal vuole che il nome di ogni programma finisca col suffisso ".p". Chiamiamo il nostro programma "inizio.p"

vi inizio.p

e produciamo il seguente testo :

```
program inizio(output);
begin
  writeln('Quando vedo questa scritta vuol dire')
  writeln('che il programma funziona')
end.
```

Ricordiamoci che in vi, quando e' stata abilitata l'autoindentazione, dopo aver dato

:se sw=<numero>

possiamo spostarci a sinistra col cursore in inserimento con il carattere

<ctrl>D

Da notare che il programma, anche se piccolo, e' stato indentato, cioe' e' stato scritto nella maniera piu' leggibile. Lo stesso programma potrebbe essere stato scritto nella seguente maniera:

```
program inizio(output);
begin writeln('Quando vedo questa scritta vuol dire')
```

```
writeln('che il programma funziona') end.
```

In questa maniera e' gia' piu' faticoso leggere il programma. Proviamo ora a compilare il programma col comando

```
pi inizio.p
```

L'output della compilazione e' il seguente

```
Tue Nov 23 18:52 1982  inizio.p:
      4      writeln('che il programma funziona')
e -----^----- Inserted ';'

```

In questo modo il compilatore ci ha avvisato che e' stato commesso un errore nella stesura del testo. L'errore in questione si riferisce al primo carattere della quarta riga, (il punto dove viene trovato l'errore e' indicato dal carattere ^) il compilatore se ne e' accorto cioe' in quel punto. E' evidente allora che l'errore si riferisce alla linea precedente ed e' legato all'inserzione di un ';'. Rivedendo il testo vediamo che manca un ";" alla fine della terza linea. L'errore indicato e' stato contrassegnato con "e". Gli errori possono essere di tre tipi :

E : e' un errore fatale che non puo' essere ignorato dal compilatore e non viene effettuata la generazione di codice per cui non e' possibile eseguire il programma.

e : e' un errore sintattico, il compilatore lo segnala e lo "aggiusta", e' possibile percio' eseguire il programma

w : sono "warning", avvisi di potenziali errori o di inutilizzo di certe variabili, non pregiudica in ogni caso la generazione del codice.

Dopo una compilazione senza errori fatali, il codice ottenuto e' un file di nome "obj", contrazione di object. L'obj del nostro programma appare cosi' nella lista

```
-rwx--x--- 1      1662 Nov 23 18:57 obj
```

Quando viene creato dal compilatore, gli vengono settati automaticamente i bit di eseguibilita'. Possiamo ora eseguire il programma dando semplicemente

```
obj
```

ed avremo

```
Quando vedo questa scritta vuol dire
che il programma funziona
```

```
2 statements executed in .02 seconds cpu time.
```

alla fine dell'esecuzione, l'interprete del codice riporta il numero di statement e il tempo di cpu. Possiamo cambiare

nome ad obj ed eseguirlo col nome nuovo.
 Una maniera per compilare ed eseguire il testo con lo stesso comando, ci e' data dal comando "pix"; con

```
pix inizio.p
```

supponendo di aver rimosso l'errore originario, otteniamo

```
Execution begins...
Quando vedo questa scritta vuol dire
che il programma funziona
Execution terminated.
```

```
2 statements executed in .03 seconds cpu time.
```

In questo modo l'obj generato viene direttamente caricato in memoria senza comparire nella directory.

1.16.2. Un programma piu' grosso

Il programma che segue stampa il grafico della funzione seno. Lo chiameremo grafico.p (vedi Fig.9). Proviamo a compilare il nostro testo

```
pi grafico.p
```

ed otteniamo la Fig.10 Ora, siccome siamo agli inizi, ci risultera' piu' comodo fare stampare il risultato della compilazione col testo incluso. Pi ha varie opzioni, l'opzione "l" lista il programma coi numeri di linea mentre compila.

```
pi -l grafico.p
```

produce la Fig.11 Possiamo allora mandare tutto cio' su stampante con l'istruzione

```
pi -l grafico.p | lpr
```

Analizziamo ora gli errori :

- 1) Abbiamo scritto readln invece della procedura predefinita readln, in questo modo il compilatore assume che sia una procedura definita dal programmatore, vede che non e' stata dichiarata all'interno del programma e segnala un errore.
- 2) Abbiamo scritto lung invece di lungn, il compilatore la vede come variabile non dichiarata.
- 3) Manca una parentesi a destra dell'espressione.

```

program graph(input,output);
{ stampa il grafico di una funzione }
const larghezza=60; lung=22;
var   a,b,c,d      :real; { estremi inferiori e superiori
                           sugli assi x e y }
      x,passo      :real;
      m,q          :real;
      i,j          :integer;

      function fun(x:real):real;
      begin fun:=sin(x) end;

begin
  readln(a,b,c,d); { estremi degli assi x e y }
  passo:=(b-a)/(lung-1);
  x:=a;
  for i:=1 to lung do
    begin
      if (fun(x)>=c) and (fun(x)<=d) then
        { verifica che il punto cada tra gli estremi }
        begin
          m:=(larghezza-1)/(d-c;
          q:=1-m*c;
          writeln('*':round(m*fun(x)+q))
        end;
      x:=x+passo
    end
  end.

```

Fig. 9 Program graph.

```

-----
Tue Nov 23 23:50 1982  grafico.p:
  14      redln(a,b,c,d); { estremi degli assi x e y }
E -----^--- Undefined procedure
  17      for i:=1 to lung  do
E -----^--- Undefined variable
  22          m:=(larghezza-1)/(d-c;
e -----^--- Inserted ')'
In program graph:
  w - variable a is used but never set
  w - variable b is used but never set
  w - variable c is used but never set
  w - variable d is used but never set
  w - variable j is neither used nor set
  E - redln undefined on line 14
  E - lung undefined on line 17

```

Fig. 10 Compilazione.

 Berkeley Pascal PI -- Version 2.0 (Fri May 15 19:29:20 1981)

Tue Nov 23 23:50 1982 grafico.p

```

1  program graph(input,output);
2  ( stampa il grafico di una funzione )
3  const larghezza=60; lung=22;
4  var   a,b,c,d      :real; ( estremi inferiori e superiori
5                                sugli assi x e y )
6      x,passo       :real;
7      m,q           :real;
8      i,j           :integer;

10     function fun(x:real):real;
11     begin fun:=sin(x) end;

13  begin
14     redln(a,b,c,d); ( estremi degli assi x e y )
E -----^----- Undefined procedure
15     passo:=(b-a)/(lung-1);
16     x:=a;
17     for i:=1 to lung do
E -----^----- Undefined variable
18     begin
19         if (fun(x)=c) and (fun(x)<=d) then
20             ( verifica che il punto cada tra gli estremi )
21             begin
22                 m:=(larghezza-1)/(d-c;
e -----^----- Inserted ')'
23                 q:=1-m*c;
24                 writeln('*':round(m*fun(x)+q))
25                 end;
26                 x:=x+passo
27             end
28     end.
```

In program graph:

```

w - variable a is used but never set
w - variable b is used but never set
w - variable c is used but never set
w - variable d is used but never set
w - variable j is neither used nor set
E - redln undefined on line 14
E - lung undefined on line 17
```

Fig. 11 Compilazione estesa.

 Conviene inoltre togliere la variabile j poiche' non viene utilizzata in nessuna parte del programma. Il testo corretto e' visibile in Fig.12. Compilandolo nuovamente non

```

-----
program graph(input,output);
( stampa il grafico di una funzione )
const larghezza=60; lungh=22;
var   a,b,c,d      :real; ( estremi inferiori e superiori
                           sugli assi x e y )
      x,passo       :real;
      m,q           :real;
      i             :integer;

      function fun(x:real):real;
      begin fun:=sin(x) end;

begin
  readln(a,b,c,d); ( estremi degli assi x e y )
  passo:=(b-a)/(lungh-1);
  x:=a;
  for i:=1 to lungh do
    begin
      if (fun(x)>=c) and (fun(x)<=d) then
        ( verifica che il punto cada tra gli estremi )
        begin
          m:=(larghezza-1)/(d-c);
          q:=1-m*c;
          writeln('*':round(m*fun(x)+q))
        end;
      x:=x+passo
    end
  end.
end.

```

Fig. 12 Program graph corretto.

avremo piu' errori e possiamo eseguire il programma con

obj

Ora l'interprete sta aspettando i nostri dati di input.
Proviamo a dargli

-5 5 a 3.2

seguiti da un <return> ed avremo

standard input: Bad data found on real read

Error in "graph"+1 near line 14.

1 statements executed in .02 seconds cpu time.

Questo e' un runtime error, un errore durante l'esecuzione, in questo caso perche' viene effettuata una lettura di numeri reali ed e' stato digitato un carattere (a) non identificabile come real. Rifacciamo l'esecuzione dando come input

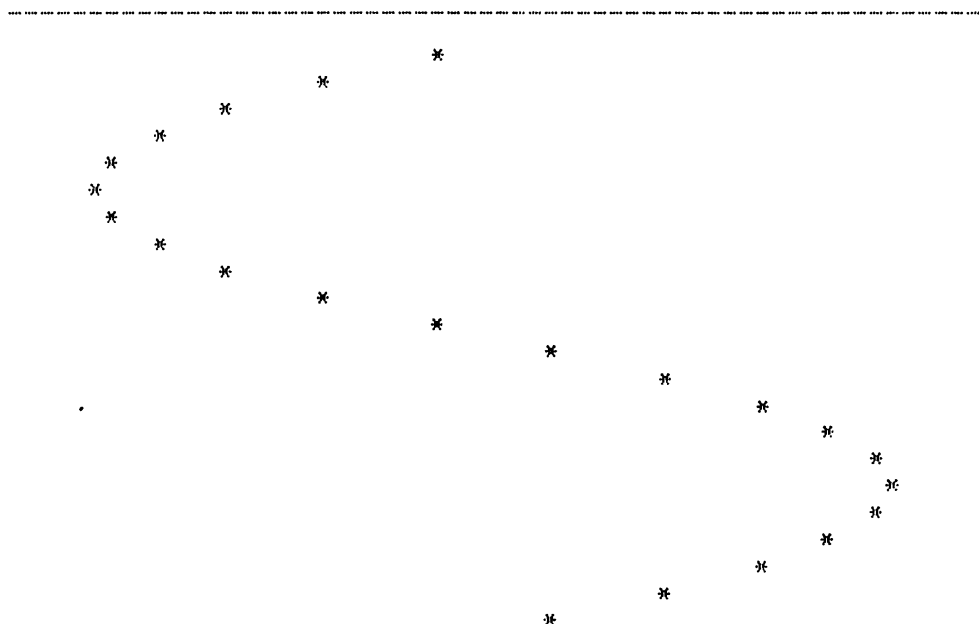
-3 3 -1.2 1.2

otterremo allora la Fig.13. Se vogliamo ottenere l'output su stampante, daremo

obj : lpr

e immediatamente dopo diano i numeri di input. Notiamo che la scritta

202 statements executed in .23 seconds cpu time.



202 statements executed in .23 seconds cpu time.

Fig. 13 Esecuzione di graph.p.

finisce su terminale invece che su stampante, poiche' e' vista come standard error. Per passare lo standard error attraverso una pipe, dare una "&" dopo la "!" (questo solo per chi ha la cshell). Nel nostro esempio sarebbe

```
obj !& lpr
```

Un'altra maniera per mandare l'esecuzione su stampante sarebbe quella di ridirigere lo standard input. Se ho creato un file chiamato input con il seguente contenuto:

```
-3 3 -1.2 1.2
```

posso mandare l'esecuzione su stampante con

```
obj < input ! lpr
```

1.16.3. Profilazione di esecuzione

Una profilazione di esecuzione consiste in una lista strutturata di un programma con le informazioni sul numero di volte in cui ogni statement e' stato eseguito in una particolare esecuzione del programma. Questo puo' essere utile per vari motivi:

1) Quando un programma termina anormalmente, per poter identificare la parte di programma che ne e' responsabile.

2) Per ottimizzare la velocita' del programma: vedere quante volte sono stati eseguiti gli statement che richiedono piu' tempo.

3) Scoprire le cause di un loop infinito.

Riferendoci ancora a grafico.p, per avere la profilazione di esecuzione, dobbiamo prima compilarlo con l'opzione "-z"

```
pi -z grafico.p
```

Ora eseguiamo

```
obj
```

A questo punto e' stato creato un file chiamato pmon.out, che servira' per avere la profilazione, che si ottiene con

```
pxp -z grafico.p
```

ricordandoci che l'output puo' essere deviato su un file o su stampante per studiarcelo meglio, otteniamo la Fig.14. Per sapere quante volte e' stato eseguito uno statement, bisogna cercare a sinistra della linea in questione la corrispondente barra verticale "I". All'inizio di questa ci sara' la corrispondente barra orizzontale con il numero di

Berkeley Pascal PXP -- Version 1.1 (May 7, 1979)

Wed Nov 24 00:22 1982 grafico.p

Profiled Wed Nov 24 15:57 1982

```

1      1.---|program graph(input, output);
( stampa il grafico di una funzione )
3      |const
3      |      larghezza = 60;
3      |      lungh = 22;
4      |var
4      |      a, b, c, d;
4      |      ( estremi inferiori e superiori
4      |              sugli assi x e y ) real;
6      |      x, passo: real;
7      |      m, q: real;
8      |      i: integer;

10     66.---|function fun(x: real): real;
11     |begin
11     |      fun := sin(x)
11     |end; ( fun )

13  1.---|begin
14  |      readln(a, b, c, d); ( estremi degli assi x e y )
15  |      passo := (b - a) / (lungh - 1);
16  |      x := a;
17  |      for i := 1 to lungh do begin
19  |  22.---|      if (fun(x) >= c) and (fun(x) <= d) then begin
22  |  22.---|      ( verifica che il punto cada tra gli
22  |              estremi )
22  |              m := (larghezza - 1) / (d - c);
23  |              q := 1 - m * c;
24  |              writeln('*': round(m.* fun(x) + q))
24  |              end;
26  |              x := x + passo
26  |      end
26  |end.

```

Fig. 14 Profilazione di esecuzione.

volte che lo statement e' stato eseguito. Per esempio, la funzione "fun" in linea 10 e' stata eseguita 66 volte.

1.16.4. Errori del compilatore

E' raro ma possibile incontrare in compilazione degli errori tipo

```
Snark in pi
```

questo vuol dire che il compilatore non e' stato in grado di "comprendere" uno statement. Cio' non vuol dire che lo statement sia sbagliato. Praticamente e' un errore del compilatore. In tal caso, bisogna capire quel'e' lo statement in questione ed ottenerlo in un'altra maniera. Uno snark possibile e' lo statement che testa l'insieme vuoto, per cui

```
if a = [] then
```

dara' uno snark. Usare molta cautela con le variabili di tipo set. Una soluzione e' rappresentata dalla funzione card (ritorna il numero di elementi presente in un set). Lo statement precedente puo' essere correttamente eseguito con

```
if card(a) = 0 then
```

1.16.5. Errori in esecuzione

Gli errori durante l'esecuzione di un programma possono essere di vari tipi. In ogni caso l'esecuzione termina in quel punto. Un possibile errore e' quello di una variabile o di un indice che va fuori dal range dichiarato. Se ho dichiarato

```
var a : array[1..20] of integer;
```

e poi ho

```
for i := 1 to 30 do j:=a[i]
```


avro' un errore eseguendo il ventunesimo ciclo del for, poiche' l'indice dell'array esce dal range 1..20. L'esecuzione allora si interrompera' in quel punto, segnalando il tipo d'errore e il numero della linea in cui e' stato trovato. Altri errori frequenti sono quelli di I/O (input/output), errori che si manifestano quando abbiamo a che fare con files esterni. Questi sono i casi piu' frequenti in cui capitano questi errori

- 1) Vogliamo leggere un file che non esiste o in cui non abbiamo permesso di lettura
- 2) Vogliamo creare un file in una directory di cui non abbiamo accesso
- 3) Chiediamo di leggere dopo la fine del file
- 4) Vogliamo scrivere su un file aperto in lettura con reset
- 5) Vogliamo leggere da un file aperto in scrittura con rewrite

Uno spiacevole errore di esecuzione e' quello di

Stack overflow

oppure di

panic : bad op code

che generalmente e' legato alla diagnostica precedente. Stack overflow indica che e' finita la parte di memoria centrale riservata all'esecuzione del nostro programma. Questo succede quando il nostro programma contiene troppe variabili od ha un codice troppo grosso. Ricordo alcuni utili consigli per ottimizzare il programma.

1) Spezzare grosse procedure in procedure piu' piccole, considerando che le variabili interne ad una procedura vengono allocate al momento del richiamo della stessa e distrutte in uscita.

2) Diminuire il numero di variabili globali, eventualmente distribuendole localmente alle procedure, se non e' possibile toglierne alcuna.

3) Le scritte (write, writeln) di stringhe occupano molto codice, cercare allora di sintetizzarle.

4) Un case statement con parecchie label occupa molta memoria, cercare allora una maniera alternativa per ottenere le stesse cose.

1.16.6. Loop infiniti

Quando ci capita che un programma non finisca mai l'esecuzione, cioè una parte di esso va in un loop infinito, l'unica maniera per interromperlo e' di trasmettergli un interrupt, nella maggior parte dei terminali attraverso il tasto <delete>. Allora per poter trovare l'errore, l'interprete segnala in che punto e' stato interrotto il programma. Particolarmente dannoso e' quando un programma va in loop mentre sta scrivendo su un file, in tal caso ci potrebbe succedere che il file assuma proporzioni tali da riempire l'intero file-system, causando disagi per tutti gli utenti.

Normalmente, se non viene altrimenti specificato (vedi la parte di procedure e funzioni predefinite), il programma termina dopo 500.000 statement eseguiti, poiche' si presume che dopo questo limite il programma sta andando in loop.

1.16.7. Eoln ed eof

I caratteri che segnalano la fine dell'input o la fine di una linea variano da macchina a macchina. Queste due funzioni evitano al programmatore di preoccuparsi del riconoscimento di questi caratteri. Eoln ed eof sono due funzioni predefinite che possono assumere valore true o false. Per UNIX il carattere di fine linea e' il carattere di newline, <return> per la tastiera. La fine dell'input viene comunicata per mezzo del carattere <ctrl>D. L'input da tastiera viene letto dal momento in cui c'e' eoln. Vediamo il piccolo programma di prova in Fig.15 per capire il funzionamento di queste due funzioni. L'esecuzione e' visibile in Fig.16. Vediamo che la scritta writeln(' letto=',ch) non avviene dopo ogni carattere digitato, ma i caratteri vengono bufferizzati e rispediti dopo un <return>. In questo modo e' possibile correggere i caratteri digitati con <back-space>.

```

program prova(input,output);
var ch :char;

begin
  while not eof do
    begin
      while not eoln do
        begin
          read(ch); writeln('  letto=',ch);
        end;
      readln
    end
  end.

```

Fig. 15 Prove su eof ed eoln.

```

d<return>
  letto=d
R<return>
  letto=R
prove<return>
  letto=p
  letto=r
  letto=o
  letto=v
  letto=e

```

(qui e' stato digitato <ctrl>D)

31 statements executed in .02 seconds cpu time.

Fig. 16 Esecuzione.

1.16.8. Reset e rewrite

Se abbiamo dichiarato *f* come file (non ci importa adesso qual'e' il tipo del file), possiamo aprirlo in lettura con

```
reset(f)
```

Se "f" e' stato dichiarato in testa al programma (assieme ad input ed output), e' sufficiente questa scrittura, altrimenti non e' sufficiente poiche' non e' stato specificato il file che vogliamo leggere e qualsiasi lettura darabbe un errore di esecuzione. Allora,

```
reset(f,'prog')
```

dove al posto di prog ci puo' essere il nome di un qualsiasi file UNIX, a patto che esista ed abbiamo i permessi per leggerlo, come

```
reset(f,'/usr/utenti/mario/routines')
```

Una volta che e' stato specificato il file, ogni successiva reset puo' essere fatta con il nome della sola variabile, anche se per chiarezza e' consigliabile dare i due nomi. Per aprire un file in scrittura possiamo dare

```
rewrite(f)
```

senza specificare altro nome, il pascal system genera un file chiamato "tmp.?" dove al posto di ? ci sara' un numero. Questo e' un file considerato temporaneo e sara' rimosso automaticamente in uscita dall'esecuzione a meno che il programma non termini anormalmente. Se vogliamo che il nostro file sia permanente, cioe' rimanga anche dopo l'esecuzione del programma, dobbiamo dare i due nomi, come

```
rewrite(f,'/lib/lib.1')
```

Il secondo nome puo' anche essere una variabile di tipo stringa.

1.16.9. Argc e argv

Abbiamo gia' visto che i comandi UNIX nella grande maggioranza dei casi accettano degli argomenti (generalmente nomi di files). Questi argomenti sono numerati da 0 in poi. Nell'istruzione

```
ls -l file1 file2 file3
```

"ls" e' l'argomento n.0

"-l" e' l'argomento n.1

"file1" e' l'argomento n.2

e cosi' via. Nell'istruzione

```
ls -l *
```

c'e' un numero di argomenti che e' uguale al numero di files contenuti nella directory + 2 per il comando e l'opzione. E' possibile dare argomenti anche richiamando l'esecuzione di un programma pascal. A questo proposito esiste una funzione `argc` e una procedura `argv` che permettono questo tipo di interfaccia con la shell. Il valore della funzione `argc` e' il numero di parametri passati al richiamo del processo. Per cui in

```
obj file1 file2
```

la funzione `argc` ha valore 3. Invece all'interno del programma un richiamo tipo

```
argv(i,nome)
```

dove `i` e' un integer e `nome` una variabile di tipo stringa, assegna l'iesimo argomento (partendo da zero) alla variabile `nome`. Nell'esempio precedente,

```
argv(1,nome)
```

assegna a `nome` la stringa "file1". Vediamo ora un programma che conta il numero di linee dei files che sono passati come argomenti. Lo chiamiamo `linee.p` (Fig.17). Dopo averlo compilato, cambiamogli nome con

```
mv obj linee
```

e proviamo

```
linee linee.p linee
```

Vogliamo cosi' contare il numero di linee nei files chiamati "linee.p" (il programma sorgente) e "linee" (il risultato della compilazione) ed otteniamo

```
linee.p :      34 linee
linee :      10 linee
```

```
5186 statements executed in 3.10 seconds cpu time.
```

Naturalmente e' piu' lento dell'istruzione "wc -l" di sistema, poiche' quest'ultima e' scritta nel linguaggio C come gran parte del sistema operativo. Il C genera un codice assembler che e' parecchio piu' veloce di un codice che deve essere interpretato da un altro programma al momento dell'esecuzione. Se diamo come argomento il nome di un file non esistente,

```

program contalinee(input,output);

var cont,i,j : integer;
    nome : packed array[1..50] of char;
    ( pathname dell'argomento )
    f : text;
begin
    i:=0;
    while not (i=argc-1) do
        begin
            i:=i+1;
            argv(i,nome);
            ( assegno a nome l'iesimo
              argomento )
            reset(f,nome);
            cont:=0;
            while not eof (f) do
                begin
                    while not eoln(f) do get(f);
                    get(f);
                    cont:=cont+1;
                    ( incremento il contatore dopo
                      che ho trovato eoln )
                end;
                j:=1;
                while not ( nome[j] = ' ' ) do
                    ( scrivo il nome del file )
                    begin
                        write(nome[j]);
                        j:=j+1
                    end;
                writeln(' :',cont,' linee');
            end
        end;
end;

```

Fig. 17 Programma con argomenti.

```

linee linee.p xxxx

```

succede questo

```

linee.p :          34 linee

```

```

Could not open xxxx: No such file or directory

```

```

Error in "contalinee"+8 near line 15.

```

1417 statements executed in .87 seconds cpu time.

l'esecuzione si ferma quando incontra l'errore.

1.16.10. Tipi standard, limiti

Il tipo integer e' definito da minint a maxint, due costanti predefinite con i valori -2147483848 e +2147483847. Il tipo char ha i 128 caratteri ASCII. Il tipo real e' implementato usando 64 bit in floating point, ha circa 17 cifre di precisione con numeri che venno da 10^{-38} a 10^{38} (leggere ^ come "elevato alla"). I set possono avere al piu' 32767 elementi. Gli identificatori sono considerati uguali tra loro quando hanno uguali i primi 160 caratteri. Sono accettati fino a 20 livelli di nesting di procedure o funzioni.

1.16.11. Opzioni di pi

Pi accetta piu' di un opzione per volta, le opzioni possono essere date al momento del richiamo del compilatore, tipo

```
pi -w prog.p
```

oppure possono essere date all'interno del programma incluse tra parentesi graffe ed il carattere dollaro a sinistra, come

```
{ $w- }
```

le opzioni date da programma devono apparire prima del program statement. "-p" toglie il conteggio del numero degli statement

"-t" disabilita i controlli in run time, da usarsi quando si e' certi che il programma funziona.

"-b" bufferizza l'output in un blocco, e' possibile fargli bufferizzare piu' di un blocco specificandolo da programma; con

```
{ $b2 }
```

l'output viene bufferizzato in 2 blocchi.
 Queste tre opzioni velocizzano l'esecuzione di un programma, per cui quando siamo certi che il programma funziona, possiamo dare

```
pi -ptb <nome-programma>
```

"-l" genera un listing del programma mentre compila
 "-s" segnala funzioni e procedure che non sono standard pascal, inoltre accetta le parole chiave e gli identificatori scritti in maiuscolo

1.16.12. Pxp

Pxp, dato senza opzioni, produce il testo di input (sempre un programma pascal) formattato (indentato). Con l'opzione "--", indenta e sottolinea le parole chiave. L'opzione già vista "-z" produce una profilazione di esecuzione. Per le altre opzioni, consultare i testi in bibliografia.

1.16.13. Cross-reference

Il comando pxref produce un cross-reference del programma, cioè genera un listing del programma e segnala le linee in cui sono utilizzate le variabili e le procedure. Dando

```
pxref linee.p
```

otteniamo la Fig.18.

1.16.14. Procedure predefinite

```
argv(i,nome)
```

Assegna l'iesimo argomento alla stringa nome

argc	9					
argv	12					
char	4					
cont	3	16	21	21	32	
contalinee	1					
eof	17					
eoln	19					
f	6	15	17	19	19	20
get	19	20				
i	3	8	9	11	11	12
input	1					
integer	3					
j	3	25	26	29	30	30
nome	4	12	15	26	29	
output	1					
reset	15					
text	6					
write	29					
writeln	32					

19 identifiers

43 occurrences

Fig. 18 Cross-reference (e' stato omissso il listing).

```
remove(a)
```

Con a di tipo stringa, rimuove il file chiamato a

```
date(a)
```

Assegna alla stringa a la data corrente nella forma "ggmmaa", due caratteri per il giorno, tre per il mese e due per l'anno

```
time(a)
```

Assegna alla stringa a l'ora attuale nella forma "hh:mm:ss"

```
stlimit(i)
```

Dove i e' un integer, causa la terminazione forzata del programma dopo i statement

1.16.15. Funzioni predefinite

`argc`

Ritorna il numero di argomenti passati all'esecuzione del programma

`card(a)`

Ritorna il numero di elementi contenuti nel set a

`random(x)`

dove x e' un real, invoca un generatore di numeri pseudo-casuali. Ritorna un numero compreso tra 0.0 e 1.0. Il numero viene generato dall'espressione.

$(\text{seed} * a + c) \bmod m$

Il seed iniziale e' 7774755

`seed(i)`

dove i e' un integer, setta il generatore di numeri pseudo-casuali seed all'integer i e ritorna il precedente seed.

`sysclock`

Ritorna il numero di millisecondi di cpu usati dall'esecuzione del programma

CAPITOLO 2

Qualcosa di piu`

2.1. I LINK

Diamo l'istruzione

```
ls -la
```

l'opzione "a" (all) del comando ls elenca tutti i files, anche quelli il cui nome comincia con un punto. Tra questi sono sempre presenti le directory "." e "..".

```
drwx--x--- 3      384 Nov 27 18:34 .  
drwx----- 6      672 Nov 27 18:36 ..  
-rw----- 1       76 Nov 27 15:05 2.con  
-rw----- 1  58309 Nov 26 18:05 CAP.1.C  
drwx--x--- 2       32 Nov 27 18:31 documento  
-rw----- 1      663 Nov 24 00:22 graph.p  
-rw----- 1      657 Nov 25 22:25 linee.p
```

Un link effettua un collegamento tra nomi simbolici dello stesso file.

Vediamo che alcuni files hanno un numero di links (secondo campo di ogni riga) maggiore di uno. In particolare le directory hanno almeno 2 link. La directory "." ha 3 link, ".." ne ha 6, "documento" ne ha 2.

Il fatto che la directory "." ha tre link, significa che le

stesse informazioni contenute nel file "." sono presenti in altri 2 pathname (il terzo e' quello presente in lista). Un link e' presente nella directory padre nella cui lista comparira' il nome della directory corrente, vediamo lo percio' con

```
ls -la ..
```

```
drwx----- 6      672 Nov 27 18:36 .
drwxr-xr-x15    272 Jan  1 01:45 ..
-rw----- 1      442 Nov 16 18:05 .cshrc
-rw----- 1    1205 Oct 13 17:03 .login
-rw----- 1      19 May 21 13:32 .logout
-rw-r--r--96    725 Oct 21 20:46 .profile
drwx--x--- 3     384 Nov 27 18:53 UNIX
drwx--x--- 2     448 Nov 24 16:23 bibliografie
drwx--x--- 5     144 Sep 16 03:32 sorgenti
-rw----- 1     878 Nov 27 18:36 lettera
drwx--x--- 2     224 Nov 26 00:02 poi
```

La current directory "." e' chiamata UNIX ; vediamo che nella lista della directory padre UNIX ha 3 link: e' fisicamente lo stesso file di ".". L'ultimo link e' rappresentato dal file ".." nella directory "documento". Per cui,

```
./
../UNIX
documento/..
```

sono lo stesso file, che compare in 3 posti diversi con 3 nomi diversi. Analogamente il file ".." ha 6 link, uno per se stesso, uno per la propria directory padre e 4 per le altre 4 directory contenute in esso : UNIX, bibliografie, sorgenti, poi.

I link tra directory vengono messi automaticamente dal sistema. E' possibile creare dei link tra files che non siano directory con l'istruzione

```
ln <file-esistente> <nuovo-riferimento>
```

Così',

```
ln ../lettera messaggio
```

effettua un collegamento tra il file "lettera" presente nella directory padre ed un file chiamato "messaggio" nella current directory. Dando

```
ls -l m*
```

otteniamo

```
-rw----- 2      878 Nov 27 18:36 messaggio
```

Anche il file ../lettera avra' adesso due link. Quando vogliamo rimuovere un file che ha piu' di un link, di fatto non viene rimosso il file ma viene solo tolto il link in quel determinato nome.

2.2. I FLAG DEI FILE

Abbiamo gia' visto il significato dei bit d,r,w,x dei files, vediamo ora altri.

Il bit s

Il bit s significa "substitute user-id on execution", ha senso solo nei file eseguibili e la sua funzione e' quella di fare in modo che un normale utente, limitatamente al momento in cui sta eseguendo il programma, assuma le stesse caratteristiche e priorita' del proprietario del file. L'esempio del comando lpr puo' spiegare molto bene il funzionamento di questo bit.

```
ls -lagp /bin/lpr
```

Produce

```
opr bin      -rwsr-x--x 1   13814 Oct 10 09:41 /bin/lpr
```

Il bit s indica che un utente, mentre esegue il comando lpr, ha le stesse possibilita' dell'operatore (proprietario di lpr). Quando viene attivato il comando lpr, parte il "daemon" di stampa il cui compito e' quello di accodare le richieste dei files da stampare nella directory /usr/spool/lpd e di fare in modo che questi files siano trasferiti su stampante.

Non e' possibile (eccetto per l'operatore) scrivere in questa directory poiche' si potrebbero cosi' rimuovere stampe di altri utenti o cambiare le priorita' di uscita della stampa. Daltronde, non possono uscire files su stampante se non passano prima da questa directory. Bisogna cioe' che un utente abbia la possibilita' di scriverci dentro. Lpr permette di

fare questa cosa solo durante l'esecuzione del comando. Il senso del bit `s` e' appunto quello di fare in modo che gli utenti possano usufruire di alcuni vantaggi durante l'esecuzione di un comando.

Un'altro esempio del funzionamento del bit `s` ci e' dato dal comando `"passwd"` che, all'atto dell'inserimento di una password, permette di modificare il file `/etc/passwd`, normalmente non scrivibile se non dal proprietario.

Il bit `t`

Il bit `t` e' presente nei comandi piu' usati del sistema, tipo `vi` o la shell stessa. Proviamo

```
ls -l /bin/sh /usr/ucb/bin/vi
```

Ed otteniamo

```
-rwxr-x--t 1 20236 Apr 17 06:50 /bin/sh
-rwxr-xr-t 5 69826 Feb 18 22:14 /usr/ucb/bin/vi
```

In entrambi compare il flag `t`. Il codice di un comando, per poter essere eseguito, dev'essere trasportato in memoria centrale. Se un file eseguibile ha le aree dati e testo separate, questo bit fa si che il testo venga mantenuto permanentemente nell'area di swap rendendo cosi' piu' veloce il caricamento in memoria centrale. L'area di swap e' quella parte di disco su cui vengono trasferiti temporaneamente i programmi quando finisce lo spazio a disposizione in memoria centrale.

2.3. LA SHELL

La shell e' un interprete di comandi. Supponiamo che quando vi collegate, la shell che interpreta ed esegue i vostri comandi sia `/bin/sh`. Nel prossimo capitolo verra' spiegata in particolare sia `/bin/sh` sia un'altra shell presente in molti sistemi: `/bin/csh`. Nel caso in cui abbiate la `csh`, per seguire in pieno questa parte, date prima l'istruzione

```
/bin/sh          o anche solo
sh
```

Le differenze tra le due non sono enormi, comunque se non sapete qual'e' la shell che state utilizzando, e' il caso che lo chiediate ad altri utenti o ad un operatore. Dal momento in cui ci siamo collegati al sistema, la shell e' in attesa di un nostro comando per interpretarlo ed eseguirlo.

2.3.1. I file di inizializzazione

Prima ancora che vi si presenti il prompt, un file di comandi chiamato `".profile"`, se esiste, e' stato interpretato ed eseguito. Questo e' un file che di solito compare attraverso links in tutte le home directory e serve per assegnare certi parametri utili alla shell. `".profile"` e' dunque un file eseguibile composto di comandi shell. Se nella vostra home directory e' presente un file chiamato `".user"`, esso verra' eseguito immediatamente dopo il `.profile`. Questo file serve per fare eseguire dei comandi al momento in cui si ci collega e per personalizzare la propria shell. Come vedremo piu' avanti, la shell puo' essere usata come vero e proprio linguaggio di programmazione. Creiamo il file `.user` con i seguenti contenuti

```
ls -l
num=`who | wc -l`
echo 'Oggi ci sono' $num 'utenti'
```

Ora dobbiamo assegnare l'eseguibilita' a `.user` con l'istruzione

```
chmod +x .user
```

Ora, dal momento in cui ci colleghiamo la prossima volta, oltre ad avere il messaggio del giorno ecc..., avremo, prima del prompt, la lista della home directory e la scritta

```
Oggi ci sono 5 utenti
```

se il numero di utenti collegati e' uguale a 5. Analizziamo riga per riga il contenuto del file ".user". La prima riga e' l'istruzione gia' conosciuta di lista estesa, la seconda riga ha questo significato : assegna alla variabile "num" l'output del comando che e' racchiuso tra apici al contrario (il carattere "`"). La terza linea contiene l'istruzione echo, che e' l'istruzione di stampa. La scritta "oggi ci sono" e' racchiusa tra apici, il che vuol dire di stamparla senza trasformazioni, \$num e' il richiamo della variabile "num", si intende cioe' di stampare il contenuto della variabile "num". Quando ci si riferisce a una variabile, a parte il caso di assegnamento alla stessa, bisogna far precedere l'identificatore dal carattere "\$". Alla fine, facciamo stampare la parola "utenti".

Il file ".user" va in esecuzione automaticamente quando ci colleghiamo, ma e' possibile eseguirlo in qualsiasi momento come qualsiasi altro file di comandi shell in una delle seguenti maniere:

```
.user
sh .user
. .user
```

Gli ultimi due modi mandano in esecuzione il file anche se non ha il bit di eseguibilita'. L'ultimo equivale a scrivere i comandi contenuti in .user da tastiera.

2.3.2. Variabili shell

Abbiamo visto che e' possibile definirsi delle variabili ed assegnare ad esse una stringa. Il contenuto di una variabile e' sempre visto come insieme di caratteri. E' presente un insieme di variabili predefinite che possono essere modificate dall'utente, per personalizzare la propria shell. Provate a questo proposito a dare l'istruzione

```
set
```

Questo visualizza il contenuto di tutte le variabili

presenti al momento, sia quelle predefinite che quelle definite dall'utente. Per avere solo le variabili predefinite, date l'istruzione

```
printenv
```

E' possibile modificare queste variabili, vediamone alcune.

EXINIT e' una variabile che inizializza alcuni comandi per gli editor di sistema. E' possibile assegnare alcuni parametri dell'editor una volta per tutte, senza che ci sia bisogno di comunicarglieli ogni volta che entriamo in editor. Per esempio, vogliamo che ogni volta che richiamiamo l'editor, abbiamo gia' assegnata l'autoindentazione, il valore di shiftwidth uguale a 3 e /bin/csh la shell che chiamiamo all'interno dell'editor. Allora diamo l'istruzione

```
EXINIT='set ai sw=3 shell=/bin/csh'
```

Questa istruzione, come tutte quelle che seguono, possiamo metterla nel file ".user" per non doverla digitare ogni volta che ci colleghiamo al sistema.

```
echo $EXINIT
```

se abbiamo fatto le cose correttamente ci da

```
set ai sw=3 shell=/bin/csh
```

HOME contiene il pathname della nostra home directory. E' possibile modificarla in modo che l'istruzione "cd" ci faccia entrare in una directory diversa da quella di partenza.

```
HOME=<pathname>
```

Ci cambia la home directory nel pathname indicato.

PATH contiene i pathname, separati da un ":", delle directory in cui vengono ricercati i comandi di sistema. La shell quando riceve un comando guarda prima se risiede nella current directory, successivamente ricerca il comando all'interno delle directory contenute in \$PATH. Se vogliamo che la shell cerchi dei comandi (in questo caso nostri programmi) contenuti in una directory chiamata "bin", interna alla nostra home directory, procediamo cosi'

```
PATH=~$HOME/bin:$PATH"
```

In questo modo inserisco all'inizio di \$PATH, il pathname di \$HOME/bin e un due punti che funge da separatore. Così, quando la shell riceve un comando, per prima cosa lo cerca nella current directory, successivamente nella directory bin della nostra home directory e infine nelle directory di sistema.

PS1 e PS2 sono i prompt di default. PS2 e' il secondo prompt quando un'istruzione avviene su piu' di una riga. L'istruzione

```
PS1='VAI ! '
```

ottiene "VAI ! " come prompt.

TERM contiene il tipo del terminale usato, per fare in modo che il visual editor vi conosca il terminale, poiche' le istruzioni di movimento del cursore, reverse ed altre, sono differenti da un terminale all'altro. Le caratteristiche del terminale sono comunque utili a tutto il sistema, non solo agli editor. Le caratteristiche dei terminali sono comunicate al sistema attraverso il file /etc/termcap. Attenzione a modificare la variabili TERM, poiche' se diamo per esempio

```
TERM=kkkk
```

Quando entriamo in vi, ci dira' che non conosce il terminale poiche' il terminale chiamato kkkk non esiste. In tal caso possiamo uscire dall'editor con "q!" e rimettere \$TERM a posto.

2.3.3. Altre variabili

Queste variabili, a parte "\$?" che viene assegnata dopo ogni comando, sono assegnate inizialmente dalla shell.

\$? contiene lo stato di uscita (status) dell'ultimo comando dato. Ogni comando dato ritorna un numero, che e' zero se il comando e' stato completato in maniera regolare, altrimenti ritorna un valore diverso da zero. Questo valore viene caricato nella variabile \$? . Se diamo

```
rm <nomefile-che-non-esiste>
```

ci verra' segnalato

```
rm : <nomefile> nonexistent
```

Diamo poi

```
echo $?
```

ed otteniamo 1 poiche' il comando precedente ha riportato un errore. Una ulteriore "echo \$?" ci riporterà a questo punto uno 0. La ragione e' semplice: il comando "echo \$?" e' stato eseguito in maniera corretta. Questa variabile e' utile quando vogliamo sapere se una certa istruzione in un programma shell, e' andata a buon termine o quando facciamo ripetere una certa serie di istruzioni finche' non vengono eseguite correttamente.

`$#` E' il numero di parametri passati ad una procedura shell.

`$$` Contiene il numero del processo della shell che si sta eseguendo. Il numero del processo della shell con cui ci colleghiamo, e' sempre lo stesso fino a quando non ci scollegiamo, cosi' come il numero del processo dell'esecuzione di un file di comandi shell e' lo stesso finche' non sono state eseguite tutte le istruzioni del file e non si torna a livello del prompt. Siccome il numero del processo e' unico, `$$` e' utile quando vogliamo creare unici file temporanei, ad esempio con

```
ps > tmp$$
< serie di istruzioni >
rm tmp$$
```

Provare "echo `$$`"

`$!` Contiene il numero del processo dell'ultimo processo che e' stato mandato in background con la `&`, puo' servire per sapere se il processo ha terminato (in tal caso `$!` non contiene niente) o per terminare l'esecuzione con

```
kill $!
```

`$-` contiene il flag con cui e' stata eseguita la shell. I flag verranno spiegati nel capitolo 3.

Le variabili definibili dall'utente possono essere di

lunghezza arbitraria e possono contenere tanti caratteri quanti sono quelli della memoria disponibile. A una variabile e' possibile anche assegnare la stringa nulla:

```
a=
```

assegna la stringa vuota alla variabile "a". Il significato speciale dei metacaratteri viene inibito con gli apici, per cui

```
a=*
```

carica in a i nomi di tutti i files presenti nella directory. Ricordiamo che l'asterisco ha significato speciale per la shell ed in particolare ha il significato di "qualsiasi sequenza di caratteri".

```
a='*'
```

carica in a solo il carattere *.. Per poter caricare piu' di una parola in una variabile, bisogna racchiudere il tutto tra doppi apici (i metacaratteri vengono in ogni caso interpretati). Così'

```
a=due parole
```

non assegna niente alla variabile a. La maniera corretta e'

```
a="due parole"
```

Possiamo prendere solo alcune parole di una variabile assegnata in questa maniera:

```
set `date`
```

ci carica l'output del comando "date" in una variabile chiamata \$@ che conterra'

```
Sun Nov 28 18:54:55 GMT+1:00 1982
```

In questa stringa ci sono 6 campi, ogni campo successivo e' separato da un spazio bianco. Possiamo riferirci a un campo qualsiasi chiamandolo con \$1, \$2, \$3 ecc... Per cui

```
echo $3 $2 $6
```

ci scrive "28 Nov 1982". Alla shell come linguaggio di programmazione arriveremo dopo aver conosciuto l'uso di altri comandi.

2.4. USO AVANZATO DI VI ED ED

Abbiamo già visto nel primo capitolo l'uso di questi due editor. Ora si assume che quella parte sia già chiara al lettore.

2.4.1. Il comando di sostituzione s

Vediamo ora in particolare le sue potenzialità. Se siamo in vi possiamo richiamare il comando digitando un :, in questo modo il cursore si posiziona sulla penultima linea del video ed è pronto ad accettare i comandi. In ed non è necessario il carattere ":". Questo è il comando più complesso e più potente dei comandi editor. Le forme tipiche di questo comando sono:

```
<indirizzo/i>s/<espressione>/<rimpiazzamento>
<indirizzo/i>s/<espressione>/<rimpiazzamento>/g
```

dove <indirizzo/i> è una o più linee in cui si vuole eseguire la sostituzione <rimpiazzamento> delle stringhe che incontra <espressione>.

La g finale significa global ed indica che la sostituzione deve avvenire in tutte le ricorrenze di <espressione> nelle linee indicate. Senza g, viene sostituita solo la prima ricorrenza di <espressione> su ogni linea. Se è omessa la parte <rimpiazzamento>, le espressioni incontrate vengono semplicemente cancellate.

Indirizzi sono

```
1,$      ( in tutto il file )
-5,+10   ( dalla quinta linea sopra la linea corrente
           alla decima sotto )
$-8,$    ( le ultime 8 linee )
```

Tipici comandi di sostituzione sono

```
1,$/s/questo/quello
s/questo/quello/g
```

Negli esempi che seguiranno, per abbreviazione, verranno tralasciati gli indirizzi, tenendo presente che lo stesso

comando puo' essere eseguito su piu' di una linea. Per delimitare i campi del comando puo' essere usato qualsiasi altro carattere a parte gli spazi ed i <return>, cosi' e' corretto dare

siquesto|quello

Ricordando che quando e' stato dato un comando di sostituzione e non e' stato ottenuto l'effetto desiderato, si puo' tornare alla situazione precedente il comando stesso con "u".

2.4.1.1. Il metacarattere .

Certi caratteri quando compaiono in <espressione> assumono un significato speciale. Per togliere tale significato, e' necessario farli precedere dal carattere \. Essi possono essere usati nei comandi di sostituzione e nelle ricerche di patterns. Il punto significa "qualsiasi singolo carattere". La ricerca

/x.y/

Trova tutte le linee dove x ed y sono separate da un singolo carattere, come

```
x+y
x*y
x.y
```

Tenendo presente che "." seleziona qualsiasi carattere, il comando

s/./;/

sostituisce il primo carattere della linea con un ";". Di solito non e' il risultato desiderato. Per cui il comando corretto per sostituire il punto con un ";" e'

s/./;/

Il punto a seconda di dove si trova assume particolari significati. Nel comando

.,\$s/././

il primo punto si riferisce alla linea corrente, il secondo e' il metacarattere che seleziona qualsiasi carattere della

linea, solo il terzo e' un vero e proprio punto. Se i metacaratteri compaiono nella parte di <rimpiazzamento> non assumono significato speciale.

2.4.1.2. Il metacarattere \$

Il metacarattere "\$" significa "la fine della linea". L'uso piu' semplice e' di aggiungere una stringa a fine linea. Se abbiamo la linea

fine frase

e vogliamo aggiungergli un punto, procederemo cosi':

s/\$/./

Se invece vogliamo aggiungere uno spazio prima di ogni "?" a fine linea, ad esempio sul seguente testo

chi?
come?

con il comando

<indirizzo>s/?\$/ ?/

otteniamo

chi ?
come ?

2.4.1.3. Il metacarattere ^

Il carattere "^" significa "inizio della linea". Per cancellare i primi tre caratteri di una (o piu') linea, possiamo dare il comando

s/^....//

I tre punti successivi a "^", come visto, stanno per "3 caratteri qualsiasi".

Per cercare una linea che inizia con .sp:

/^\.sp/

Il \ e' necessario per togliere il significato speciale al punto.

2.4.1.4. Il metacarattere *

Si suppone di avere una linea che si presenta cosi' :

Citta'-----Provincia

e si vuole sostituire tutti i meno con un unico carattere bianco. Si potrebbe dare

s/-----/ /

Cio' e' piuttosto noioso. Il carattere * ci viene in aiuto. Quando preceduto da un altro carattere, ha il significato di "qualsiasi ripetizione (anche zero) del carattere precedente ". Allora il comando di sostituzione sara'

s/'-*P/' P/

Si possono combinare metacaratteri tra di loro per ottenere certi effetti. Ora si suppone di voler rimpiazzare il testo

Nome Via Citta' Provincia CAP Codice Fiscale

In

Nome Codice Fiscale

Il comando e' allora

s/e.*Co/e Co/

I due caratteri ".*" significano per "qualsiasi ripetizione di qualsiasi carattere". I metacaratteri "." e "*" sono cosi' contemporaneamente attivi.

Volendo togliere tutti i punti e i meno dalla seguente linea:

I comandi 3

possiamo dare il comando

```
s/i .* 3/i 3/
```

Il punto, se presente in <espressione>, seleziona anche i punti presenti nella linea.

Volendo invece togliere gli asterischi dalla linea

```
SERIE DI***** ASTERISCHI
```

Dobbiamo stare attenti a cio': il comando

```
s/I** A/I A/
```

Ci cambia la linea in

```
SERIE DI*****I ASTERISCHI
```

Cercate di capire perche'.

Il comando corretto e'

```
s/I\** A/I A/
```

poiche' il primo asterisco si riferisce al carattere stesso (e' stato tolto il significato speciale con \) ed il secondo ad una ripetizione del primo.

Il comando

```
s/a*/A/g
```

applicato alla linea

```
12345678
```

produce

```
A1A2A3A4A5A6A7A8
```

poiche' l'asterisco raggiunge anche zero ripetizioni di "a". Per incontrare solo una o piu' ripetizioni di "a", diamo il comando

```
s/aa*/A/g
```

2.4.1.5. I metacaratteri []

Le parantesi quadre vengono utilizzate per racchiudere un insieme di caratteri. Per togliere tutti i caratteri maiuscoli da un testo, possiamo dare i comandi

```
s/A//g
s/B//g
s/C//g
```

e cosi' via. Tutte queste istruzioni vengono riassunte da quest'unica:

```
s/[A-Z]//g
```

All'interno di parentesi quadre puo' comparire qualsiasi carattere con queste avvertenze:

- 1) Il carattere "]" deve comparire come primo carattere
- 2) Il carattere \ non ha un significato speciale
- 3) Un "^" come primo carattere significa "nessuno dei caratteri seguenti"

Il comando

```
s/[^A-Za-z]//g
```

Cancella tutti i caratteri che non siano alfabetici.

2.4.1.6. Il metacarattere &

Questo metacarattere e' soprattutto usato per evitare di digitare delle parole. Il suo significato e' "cio' che seleziona l'espressione". Per sostituire la linea

parola aggiungere

in

parola da aggiungere

possiamo dare

```
s/parola/parola da/
```

E' certamente piu' veloce dare

```
s/parola/& da/
```

Per mettere intere linee tra apici, possiamo dare

```
s/.*/'&'/
```

dal momento in cui cio' che viene selezionato da <espressione>, in questo caso l'intera linea, e' ripetuto da "&".

2.4.1.7. Sostituire a capo

Questo comando e' accessibile solo dall'editor ed. Tramite il carattere \ si puo' scomporre una linea in piu' linee. Il \ serve in questo caso a togliere il significato speciale del <return>. Se si vuole inserire una linea di asterischi prima di ogni linea che comincia con una cifra qualsiasi, si da su piu' linee il comando

```
s/^[0-9]/\
*****\
&
/
```

2.4.1.8. Modificare linee

Abbiamo visto che il metacarattere "&" sostituisce cio' che seleziona l'espressione. E' possibile inoltre sostituire parti di cio' che viene selezionato specificando in <rimpiazzamento> a quale parte si fa riferimento. Ogni parte viene delimitata da

```
\(<espressione>\)
```

Per togliere la terza parola (il delimitatore tra una parola e l'altra e' uno spazio) di una linea, si procede cosi':

```
s/\(.*\) \(.*\) \(.*\)\/\1 \2/
```

Le tre ripetizioni di `\(.*\)` seguite da uno spazio significano "qualsiasi ripetizione di qualsiasi carattere seguite da uno spazio", cioe' una parola. La parte `<rimpiazzamento>` ricopia cio' che e' stato trovato all'interno del primo (con `\1`) e del secondo (con `\2`) `\(\)`. Il terzo viene omesso dunque e' cancellato.

2.4.2. Shell in editor

E' possibile dare uno o piu' comandi del sistema operativo all'interno degli editor.

Comando unico:

- in vi si ottiene dando la sequenza di caratteri `:! "` seguiti dal comando opportuno, come in

```
:!date
```

Alla fine dell'esecuzione del comando la shell ritorna il controllo a vi che chiede un `<return>` per proseguire. All'interno del comando, i caratteri `"%"` e `"!"` assumono un significato speciale. Il carattere `"%"` viene sostituito con il nome del file su cui si sta lavorando. Così'

```
:!pr % !lpr
```

trasferisce il contenuto del file su stampante. Il carattere `"!"` viene sostituito con il precedente comando shell.

- in ed si ottiene con `!<comando>`. Non avvengono sostituzioni di `"%"` e `"!"` e la fine dell'esecuzione viene segnalata con una riga contenente `"!"`.

Piu' comandi in shell:

- in vi si ottiene con `!:sh`; alla fine del gruppo di comandi si ritorna in editor trasmettendo il carattere `<ctrl>D`. L'editor vi a questo punto chiede un `<return>`.

- in ed si ottiene con "!sh"; come per l'editor vi il gruppo di comandi viene terminato con <ctrl>D.

2.4.3. Ulteriori caratteristiche di vi

Cio' che segue riguarda esclusivamente l'editor vi. Vi e' un editor modulare, composto da piu' parti unite tra di loro. Per questa caratteristica c'e' la possibilita' che il sistema UNIX con cui lavorate abbia l'editor vi configurato diversamente.

E' possibile immettere l'output di un comando del sistema operativo all'interno del file con cui abbiamo chiamato vi con

```
:r !<comando>
```

L'output viene inserito dopo la linea in cui si trova il cursore.

Un'altra caratteristica utile e' quella di poter intervenire sul file stesso tramite un comando del sistema operativo. Cio' si ottiene con

```
<numero>!!<comando>
```

Dove <numero> sta per il numero di linee su cui si vuole fare intervenire il comando. Dopo aver digitato il secondo "!", il cursore si sposta in penultima linea ed e' pronto ad accettare il comando. Il cursore si sposta poi in prima colonna finche' non e' terminato il processo attivato.

Esempio: tramite il comando sort si vogliono ordinare le linee che vanno dalla cinquantesima alla centesima. Si potrebbe (una delle possibilita')

- 1) uscire dall'editor
- 2) attivare il comando sort ridirigendo l'output su un file temporaneo.
- 3) entrare in editor sul file ordinato togliendo tutte le linee che non ci interessano
- 4) entrare in editor sul file originale, inserire il file temporaneo e cancellare le linee non ordinate.
- 5) rimuovere il file temporaneo

L'intera procedura e' ottenibile direttamente in vi posizionandosi col cursore sulla cinquantesima linea dando

il comando

```
50!!sort
```

Oppure possiamo stampare solo alcune linee di testo con

```
<numero>!!lpr
```

Alla fine del processo di stampa, le linee indicate vengono cancellate dal file. In genere non e' cio' che si vuole. Il comando "u" ci riporta le linee in questione.

2.5. PREPARAZIONE DI DOCUMENTI

Ogni giorno vengono prodotte milioni di parole in testi. Le parole stampate vengono utilizzate nei quotidiani, nelle riviste, nei libri, nei cataloghi, nei manuali e in tutte le applicazioni dove si rende necessaria una organizzazione prima della pubblicazione o della visione. E' chiaramente impensabile organizzare a mano questa mole di parole. Grazie all'avvento dei computer e dei programmi chiamati text processors (chiamati anche word processors) questo noioso lavoro si trasforma in un compito relativamente semplice e veloce. Il text processor consente un preciso lavoro di organizzazione dei testi. Il risultato finale e' quello di avere un testo ordinato ed estremamente leggibile.

Che cos'e' possibile fare con un text processors ? La funzione piu' evidente e' quella di ottenere un testo con i margini allineati. In altre parole se il testo deve avere due centimetri di margine con il numero di pagina centrato in fondo ad ogni pagina ed anche un titolo particolare in testa ad ognuna, il processor tramite alcune direttive provvedera' automaticamente a produrre un testo conforme a tali caratteristiche.

Per comprendere meglio alcuni aspetti nell'uso del text processors, verranno forniti alcuni esempi utilizzando il set di comandi di nroff. Nel text processors nroff viene utilizzato il metodo di coniugare comandi e testo stesso in un unico file. Per comunicare un particolare comando e' necessario cominciare la linea con un punto in prima colonna seguito dall'identificatore del comando. Inoltre sono gia' predefiniti alcuni parametri di default pertanto un testo non contenente direttive viene in ogni caso formattato. Tali parametri si riferiscono alla larghezza della pagina, alla lunghezza, all'indentazione ecc... Consigliamo all'utilizzatore del processor di tener sempre presenti i valori dei predetti parametri in modo da evitare di ottenere risultati non voluti.

UNIX fornisce un ambiente ideale per preparare testi, manuali, tesi di laurea e tutto cio' che concerne documenti. Questo stesso libro e' stato creato utilizzando pochi comandi del formattatore di testi nroff. Non e' questa la

sede per spiegare in maniera particolareggiata tutti i comandi e le opzioni di nroff, si rimanda per questo ai manuali reperibili sull'argomento.

Il text processor troff e' progettato per stampanti di tipo grafico e per fotocomposizione; i comandi di troff differiscono per alcuni particolari da quelli di nroff.

Ora seguiranno alcuni piccoli esempi dell'uso di nroff con i comandi di -me. Questo comando ri richiama con

```
nroff -me <nomefile>
```

Uno dei comandi piu' semplici e' quello che serve a centrare le linee: cio' si ottiene con ".ce". Le linee

```
.ce
Frase da centrare
```

producono

```
Frase da centrare
```

Per determinare la lunghezza della riga si usa il comando ".ll [+-] <numero>". Per default la lunghezza della riga e' di 60 colonne. Esempio:

```
.ll 50      Ora la lunghezza della riga e' di 50
.ll +20     Ora la lunghezza della riga e' di 70
.ll -10     Ora la lunghezza della riga e' di 60
```

Per determinare la marginatura sinistra del testo si puo' utilizzare il comando ".ba [+-] <numero>". I titoli e note a pie' pagina non sono influenzate da questo parametro. Esempio:

```
.ba 10      Lascia 10 spazi prima di ogni riga
```

Per indentare una o piu' linee si usano i comandi ".ti +<numero>" (indentazione temporanea) ".in [+-] <numero>" (indentazione di tutto il testo fino a che non e' altrimenti specificato).

Otteniamo l'intestazione all'inizio di ogni pagina con

```
.he 's'c'd'
```

dove s sta per sinistra, c per centro e d per destra. Esempio n.1:

.he 'Nroff questo sconosciuto''Capitolo 1'

Cio' produce la scritta

Nroff questo sconosciuto Capitolo 1

all'inizio di ogni pagina.

Inoltre, se nel titolo compare il carattere "%", esso viene sostituito dal numero di pagina progressivo.

.he 'Nroff questo sconosciuto'Pagina %'Capitolo 1'

Cio' produce la scritta

Nroff questo sconosciuto Pagina 1 Capitolo 1

L'inizio di un nuovo paragrafo si comunica con ".sh <numero> "<titolo>"". <numero> indica il livello interno del paragrafo.

```
.sh 1 "L'automobile"
  <testo> ...
.sh 2 "Il motore"
  <testo> ...
.sh 2 "La carrozzeria"
  <testo> ...
.sh 3 "I colori"
  <testo> ...
.sh 3 "Le portiere"
  <testo> ...
.sh 1 "Le motociclette"
  <testo> ...
.sh 3 "I freni"
```

Avremo come output (assumiamo che tra ogni riga ci sia il testo):

```
1. L'automobile
1.1 Il motore
1.2 La carrozzeria
1.2.1 I colori
1.2.2 Le portiere
2. Le motociclette
2.1.1 I freni
```

Qualora si presentasse la necessita' di avere un indice, nroff permette la gestione automatica di cio'. L'unica cosa da comunicargli e' il nome della voce da inserire nell'indice con l'istruzione ".(x" seguita da una linea contenente la voce e un'altra contenente ".)x".

A volte e' necessario mantenere un testo cosi' come si trova (tabelle, schemi ecc ..). Le righe interessate devono trovarsi tra le direttive

```
.(1
<testo da lasciare invariato>
.)1
```

Per avere una o piu' linee vuote usiamo l'istruzione ".sp <numero>" dove <numero> indica quante linee devono essere saltate.

Per andare a nuova pagina si usa ".bp".

Quando vogliamo fare andare a capo il testo utilizziamo il comando ".br".

E' possibile definire un insieme di comandi che devono essere ripetuti sequenzialmente richiamandoli con un unico comando definito dall'utente (macro). Cio' e' permesso dal comando ".de". La fine del gruppo di istruzioni deve essere segnalata da una linea contenente "..". Esempio:

```
.de rn
.bp
.sp 5
.ti 5
..
```

Ogni volta che il processor incontra ".rn" sostituisce ad esso le 3 istruzioni (salto pagina, 5 linee vuote e temporanea indentazione di 5 spazi).

Vediamo un piccolo esempio di formattazione di alcune righe.

```
.ce
ESEMPIO DI UTILIZZO DI NROFF
.sp 2
.ti 5
.ll 55
Questo esempio dimostra come i comandi e il testo
possono essere
coniugati per essere trattati da un
word processor.
.sp
.in 10
.ll 30
Ora la lunghezza della linea e' di 30
caratteri. Questo parametro puo' essere
```

utilizzato per ottenere piacevoli effetti
come questo.

```
.in 0
```

```
.ll 55
```

```
.sp
```

```
.ti 5
```

Ora la lunghezza della linea e il margine sinistro
sono lunghi 55 caratteri.

```
.sp
```

Quando dobbiamo lasciare invariato il testo

```
.(1
```

```

=====>
|                                     >
|                                     >
|                                     >
=====>

```

```
.)1
```

Questo e' un piccolo esempio ma
significativo di come utilizzando
un minimo set di comandi di nroff sia
possibile ottenere ottimi risultati.

Il testo sovrastante produce:

ESEMPIO DI UTILIZZO DI NROFF

Questo esempio dimostra come i comandi e il testo possono essere coniugati per essere trattati da un word processor.

```
Ora la lunghezza
della linea e' di 30
caratteri. Questo
parametro puo'
essere utilizzato
per ottenere
piacevoli effetti
come questo.
```

Ora la lunghezza della linea e il margine sinistro sono lunghi 55 caratteri.

Quando dobbiamo lasciare invariato il testo

```
=====>
|                                     >
|                                     >
|                                     >
=====>
```

Questo e' un piccolo esempio ma significativo di come utilizzando un minimo set di comandi di nroff sia possibile ottenere ottimi risultati.

Troff e' un programma di formattazione testi per sistemi grafici e fotocomposizioni. E' particolarmente utile nella produzione di testi matematici e scientifici, ed in generale per ottenere testi con un'alta qualita' grafica. Non ci soffermeremo ora nel particolare dei comandi di troff, ma esporremo quelle che sono le caratteristiche piu' evidenti.

Normalmente le fotocompositrici hanno 4 serie complete di caratteri contenente romani, italici e neretto, alfabeto greco e un certo numero di caratteri speciali e simboli matematici (sommatoria, integrale, ecc..).

Troff, oltre ai comandi di formattazione gia' visti (margini, indentazione, titolazione ecc..), permette il completo controllo sulle serie di caratteri, le loro dimensioni e le posizioni degli stessi.

Troff va usato in collegamento (attraverso una pipe) con altri programmi Il programma eqn fornisce un linguaggio per fotocomporre formule matematiche; tbl permette di produrre tabelle di arbitraria complessita'.

CAPITOLO 3

I comandi

3. I comandi

In questo capitolo viene data una spiegazione dettagliata dei comandi e delle loro opzioni, corredati da alcuni significativi esempi. Vedremo per esteso comandi già usati e comandi nuovi. A causa delle molteplici configurazioni di UNIX e dell'alto numero di comandi, non verranno analizzati tutti i programmi di utilità che può contenere il sistema, ma tutti i comandi principali. La trattazione dei comandi sarà suddivisa in

NOME
SINTASSI
FUNZIONE
ESEMPI
IMPERFEZIONI

Con il primo esempio si capirà subito il significato di questa suddivisione. Nella SINTASSI, una scrittura tra parentesi quadre indica che quella parte è opzionale; una serie di tre punti indica che la parte precedente può essere ripetuta. A questo proposito,

```
cat [ -u ] file ...
```

significa che `-u` può essere omissa e la parte `file` può essere ripetuta. Non sempre nei comandi ci saranno tutte 5 le parti, così come potranno essercene altre diverse. La struttura e l'impostazione della spiegazione dei comandi è stata volutamente trattata in maniera simile a quella dei manuali ottenibili con il comando `man`, per facilitare la comprensione di entrambi.

3.1. CAT, PR, MORE

Questi tre programmi servono per visualizzare ed impaginare i files, trasferendo le informazioni contenute nel file su terminale (o altrove se richiesto). Se all'interno del file ci sono sequenze di caratteri di controllo che hanno significato per il vostro terminale (movimento del cursore, pulizia del video, ecc ...), esso rispondera' con la funzione corrispondente.

NOME

cat - concatena e stampa files

SINTASSI

cat [-u] file ...

FUNZIONE

Cat legge ogni file nella sequenza data e lo scrive nello standard output. Se non vengono dati argomenti, cat legge da standard input. In questo caso l'input termina quando viene digitato <ctrl>D, come per tutti gli altri comandi. <ctrl>D e' il carattere di end of file. L'output viene bufferizzato in blocchi di 512 byte finche' non e' presente l'opzione -u, in tal caso viene bufferizzata una linea per volta.

ESEMPLI

1) cat a b c

Visualizza, uno di seguito all'altro, i file a, b, c.

2) cat a b > c

Concatena il file a ed il file b in uno stesso file c.

IMPERFEZIONI

C'e' il rischio di perdere il contenuto di files dando istruzioni tipo

```
cat a b > a
```

poiche' il file a viene ricreato prima di essere letto.

NOME

pr - stampa ed impagina files

SINTASSI

pr [opzioni] [file] ...

FUNZIONE

Pr produce un'impaginazione dei files specificati. In output riporta i files separati da pagine intestate con la data, il nome del file od altro se altrimenti specificato ed il numero progressivo della pagina. Se non sono stati dati argomenti, pr legge da standard input. E' possibile dare le seguenti opzioni:

- n Produce un output su n colonne. Serve generalmente per mandare il file su stampante riducendo il numero di righe stampate. Da usarsi quando il file ha righe brevi.
- +n Inizia la stampa alla pagina n.
- h Seguito da un argomento, viene assunto come intestazione di ogni pagina al posto del nome del file.
- ln Assume n linee come lunghezza di ogni pagina al posto delle 66 di default.
- t Non stampa le linee di intestazione di ogni pagina. Sopprime il titolo.
- sc Quando e' richiesto un output su piu' colonne, separa le colonne tra di loro con il carattere c al posto del corretto numero di spazi bianchi. Se viene omesso il carattere c, viene assunto il carattere tab come separatore.
- m Stampa tutti i files simultaneamente, ognuno in una colonna.

ESEMPI

1) pr -2 -h TEST0 file1 file2

Stampa su due colonne prima file1 poi file2, scrivendo la stringa TEST0 come intestazione centrale di ogni pagina.

2) pr +5 -140 file1

Stampa il file file1 iniziando dalla quinta pagina assumendo 40 righe per ognuna.

NOME

more, page - visualizza files

SINTASSI

```
more [ -d ] [ -f ] [ -l ] [ -n ] [ +numero-linea ] [
+/pattern ] [ file ... ]
```

FUNZIONE

More e' un comando che permette di visualizzare un file fermandosi a seconda delle richieste dell'utente. Normalmente more si ferma dopo ogni pagina di video, stampando alla fine del video la scritta --More-- e la percentuale dei caratteri che sono stati visualizzati in rapporto a tutto il file. Se l'utente preme <return>, viene visualizzata un'ulteriore linea. Se viene premuto uno spazio bianco, viene visualizzata un'altra pagina video.

Le opzioni sono le seguenti :

-n Con n di tipo intero, definisce il numero di righe della finestra da utilizzare al posto di quelle di default. Ogni spazio bianco allora visualizza n linee.

-f Con questa opzione vengono considerate linee logiche (una linea logica finisce con il carattere di <return> e puo' occupare piu' di una linea fisica del terminale) al posto di linee video.

-l Non tratta il carattere <ctrl>L in maniera speciale. <ctrl>L e' il carattere di form feed, salta alla pagina successiva. Se questa opzione non viene data, l'output si fermerà dopo ogni linea contenente <ctrl>L.

+numerolinea

More incomincia alla linea indicata.

+/pattern

More incomincia due linee sopra la linea che contiene il pattern (e' un'espressione formata da una sequenza di caratteri che identifica una o piu' parole da ricercare) indicato.

More legge le caratteristiche del terminale dal file /etc/termcap. In un terminale capace di visualizzare 24 linee, la finestra e' di 22 linee.

Il programma more ed il programma page sono collegati attraverso link. Se il programma e' chiamato come

"page", allora lo schermo del video viene cancellato prima che venga visualizzata ogni pagina video. Inoltre vengono stampate $k-1$ linee al posto di $k-2$, dove k e' il numero di linee che contiene il video del terminale.

E' possibile dare istruzioni quando more si ferma, ed il loro effetto e' il seguente (i e' un intero opzionale che per default assume valore 1).

i <space>
visualizza altre i linee o una intera pagina video se i viene omissso.

<ctrl>D
visualizza altre 11 linee. Se e' stato precedentemente dato un valore ad i , il numero di linee e' uguale ad i .

d Come <ctrl>D

iz Lo stesso effetto che digitare uno spazio con la variante che i , se presente, diventa la nuova lunghezza della finestra.

is Salta i linee.

if Salta i pagine video.

q Esce dal more

Q Come q

= Visualizza il numero di linea a cui siamo arrivati

v Entra nell'editor vi sulla linea corrente

h Visualizza un file contenente i comandi del more con il loro significato.

i /expr
Cerca la i -esima ricorrenza del pattern expr. Se ci sono meno di i ricorrenze, la posizione nel file rimane immutata. Se e' stata trovata la ricorrenza, viene visualizzata una pagina che inizia due linee sopra la linea che contiene expr. Il <back-space> puo' essere usato per correggere l'espressione da cercare.

in Cerca la i -esima ricorrenza dell'ultima espressione cercata.

' Ritorna al punto del file in cui e' stata fatta l'ultima ricerca. Se non sono state fatte

precedenti ricerche, ritorna all'inizio del file.

!comando

Esegue un comando in shell. I caratteri "%" e "!" sono espansi rispettivamente con il nome del file su cui si sta agendo e con il precedente comando shell dato. Se vogliamo mandare in stampa il file che stiamo osservando, diamo

```
pr % ! lpr
```

Per togliere il significato speciale di "%" e di "!", li facciamo precedere da \.

i:n Salta all'i-esimo file successivo nell'elenco dei file Esempio: 2:n passa al secondo file successivo.

i:p Salta all'i-esimo file precedente. Se c'è un unico file specificato, questa istruzione salta all'inizio del file. Se l'input non è un file, viene emesso un beep e non succede nient'altro.

:f Visualizza il nome del file e il numero della linea in cui ci si trova.

. Il punto ripete il comando precedente.

I comandi partono senza bisogno di dare <return>.

ESEMPI

1) sort -n file | more -8

Filtra l'output del comando sort assegnando ad 8 linee la lunghezza della pagina video. Essendo passato attraverso una pipe, more non segnala la percentuale del file visto rispetto a tutto il file.

3.2. CONTENUTO DI DIRECTORY : LS

NOME

ls - elenca il contenuto delle directory

SINTASSI

```
ls [ -abcdfgilmqrstux1CFR ] [ file ... ]
l [ opzioni ls ] [ file ... ]
```

FUNZIONE

Senza argomenti, ls elenca il contenuto della directory corrente. Ls puo' avere come argomento o un file o una directory, nel primo caso ripete il nome del file ed eventuali altre informazioni richieste, nel secondo il contenuto della directory. Se non e' altrimenti specificato, l'output e' ordinato alfabeticamente. Queste opzioni possono essere da sole o combinate tra di loro.

- l Elenca in formato esteso, riportando i bit di protezione (mode), numero di links, numero di byte occupati e la data di ultima modifica.
- t Ordina l'elenco per data di ultima modifica, mettendo per primo il file modificato piu' recentemente.
- a Elenca tutti i file della directory, anche "." e ".." che generalmente sono esclusi.
- s Da' per ogni file l'occupazione in numero di blocchi.
- d Con una directory come argomento, elenca solamente il file directory e non i files contenuti in esso. E' usato di solito con l'opzione -l.
- u Ordina i files secondo la data di ultimo accesso. Per accesso si intende sia leggere un file (con cat, more o lpr ecc..) che modificarne il contenuto (in vi o tramite un altro programma).
- c Ordina secondo la data di creazione dei files.
- r Inverte il senso di ordinamento specificato da altre opzioni.
- i Visualizza l'i-number corrispondente ad ogni file.
- p Visualizza il proprietario di ogni file.
- g Visualizza il gruppo di appartenenza del proprietario.
- F Contrassegna le directory con il carattere "/" ed i file eseguibili con "*".
- R Genera ricorsivamente l'elenco di tutte le sottodirectory incontrate.

ESEMPI

1) ls -lagpsiF

Questa e' uno degli elenchi piu' completi che si possa ottenere. E' senza argomenti, per cui elenca il contenuto della directory corrente e produce

```
total 95
ada si 2817 1 drwx--x--- 4      384 Dec  1 19:11 ./
ada si 100  2 drwx----- 5      672 Nov 30 23:57 ../
ada si 277  1 drwx--x--- 2       96 Dec  1 16:25 1/
ada si 758 40 -rw----- 1    20257 Dec  2 17:14 2.con
ada si 319 43 -rw----- 1    22003 Dec  1 20:15 CAP.2
ada si 296  1 drwx--x--- 2      128 Dec  1 20:00 doc/
ada si 2643 3 -rw----- 1    1195 Nov 30 21:26 nroff
ada si 295  1 -rwx--x--- 1      404 Nov 29 16:40 ord*
ada si 2615 2 -rw----- 2      963 Nov 29 16:46 som
ada si 2615 2 -rw----- 2      963 Nov 28 19:41 som2
```

I files sono ordinati in ordine alfabetico poiche' non e' stato specificato diversamente, il primo campo e' la login del proprietario del file (opzione p), il secondo il gruppo di appartenenza (g), il terzo l'i-number (i), il quarto il numero di blocchi occupati dal file (s), il quinto i flag di protezione del file (l), il sesto il numero di link (l), il settimo il numero di byte (l), l'ottavo la data di ultima modifica (l), il nono il nome del file (sempre presente anche senza opzioni) e il decimo il carattere "/" o "*" (F).

I nomi som e som2 sono lo stesso file, uniti tramite un link. Da notare che hanno lo stesso i-number.

3.3. RM

NOME

rm, rmdir - rimuove files o toglie link

SINTASSI

```
rm [ -fri ] file ...
rmdir directory .
```

FUNZIONE

Rm rimuove uno o piu' files da una directory. Se un file ha piu' di un link, rm toglie il link a quel file. Per poter rimuovere un file e' necessario avere permesso di scrittura nella directory mentre non e' necessario avere permessi di lettura o scrittura nel

file stesso.

Se un file non ha il permesso di scrittura e lo standard input e' il terminale, viene visualizzato il numero che corrisponde alle protezioni e viene letta una linea da terminale. Se questa linea inizia con y il file viene cancellato, altrimenti rimane.

Rmdir rimuove una directory solo se non contiene nessun file.

Queste opzioni possono essere combinate tra loro :

- f Forza la cancellazione di un file anche se non esiste il permesso di scrittura. Se la directory non e' accessibile in scrittura dall'utente, il file non viene in ogni caso cancellato.
- r L'argomento o gli argomenti di questa opzione devono essere directory. Rimuove ricorsivamente tutti i file di tutte le directory e sottodirectory incontrate nel cammino. Serve inoltre per rimuovere una directory con tutti i suoi contenuti anche se non vuota. L'istruzione (non verra' mai eseguita da nessuno) `rm -r /`, rimuove tutto il file system, praticamente annulla il sistema.
- i Per ogni file da cancellare, viene chiesta conferma su terminale, se la risposta inizia con y, il file viene cancellato. Combinata con l'opzione -r, chiede la conferma per ogni directory.

DIAGNOSTICHE

Generalmente si spiegano da se'. Non e' possibile rimuovere il file `~..` per impedire eventuali grossi guai per se stessi e per altri utenti in un'istruzione tipo `"rm -r ~."`.

3.4. CREAZIONE DI DIRECTORY

NOME

`mkdir` - crea una directory

SINTASSI

`mkdir < nome-directory > ...`

FUNZIONE

Mkdir crea le directory specificate con il mode 777 (leggibile, scrivibile ed eseguibile a tutti) a meno che non abbiamo assegnato un mode differente con "umask". I files "." e ".." sono creati automaticamente al momento di creazione della directory.

Per eseguire mkdir e' necessario avere il permesso di scrittura nella directory.

3.5. COPIE: CP

NOME

cp - copia

SINTASSI

```
cp file1 file2
cp file ... directory
cp file directory/nome
```

FUNZIONE

Nella prima forma, file1 e' copiato in file2. I flag di protezione e le proprieta' di file2 rimangono invariate se file2 esiste gia', altrimenti assume le stesse informazione di file1.

Nella seconda forma, vengono copiati uno o piu' files nella directory indicata, mantenendo i nomi originali.

Nella terza forma viene copiato il file nella directory indicata con il nuovo nome.

MESSAGGI DI ERRORE

Cp non consente di copiare un file su se stesso.

3.6. MV

NOME

mv - sposta o cambia nome a files o directory

SINTASSI

```
mv file1 file2
mv file ... directory
mv file directory/nome
```


FUNZIONE

Nella prima forma, mv cambia il nome di file1 in file2. Se file2 e' gia' esistente viene rimosso prima che file1 cambi nome. Se file2 non ha il permesso di scrittura, mv visualizza il mode e legge una linea da input (come per il comando rm). Se la linea inizia con y, viene eseguito il cambio di nome.

Nella seconda forma, i files elencati vengono spostati nella directory indicata con i nomi originali.

Nella terza forma il file viene spostato nella directory indicata con il nome nuovo.

-f Forza mv anche se file2 non ha permesso in scrittura

ESEMPI

1) mv *.p programmi

Sposta tutti i files che finiscono in .p nella directory programmi con il nome originario.

IMPERFEZIONI

Se file1 e file2 si trovano sotto differenti file-system, mv deve prima copiare il file e poi rimuovere l'originale. In questo caso il proprietario diventa quello del processo di copia e vengono persi eventuali link ad altri files.

3.7. PASSWORD

Il file /etc/passwd viene letto da numerosissimi comandi per ricercare password, proprietari di files, nomi di utenti, ecc... Alcuni di questi programmi sono "passwd", "login", "lpr".

Ogni linea di /etc/passwd corrisponde ad un utente del sistema. Ogni utente e' diviso in 7 campi separati da un ":". Questo e' il significato dei vari campi :

- 1 nome della login
- 2 password in codice (il codice varia di volta in volta e non e' conosciuto da nessuno).
- 3 numero della user id associato alla login

- 4 numero del group id (il nome del gruppo e' reperibile nel file /etc/group)
- 5 nome reale della persona, ufficio di lavoro od altro
- 6 home directory
- 7 interprete di comandi da eseguire. Se e' vuoto, per default viene assunto /bin/sh

Questo e' un esempio di 2 utenti UNIX in /etc/passwd

```
opr:06V9n9jTxn8ac:0:0:Super User:/priv/opr:/bin/csh
sara:NbvzFflY:220:12:Sara,Carlo:/usr/utenti/corso/sara:
```

E' utile che ogni utente abbia una password che dia accesso completo ai propri files. Questo sia per proteggere i propri files da altri utenti sia per evitare che per errore qualcuno si colleghi con il nostro codice e faccia dei pasticci poiche' in sistemi che supportano molti utenti (corsi universitari ecc...) le login di accesso sono, per semplicita' di gestione del sistema, molto simili tra loro. Questo non vuol dire che i propri files debbano essere stretti gelosamente dal possessore, decidera' poi lui chi e come avra' accesso ai propri files (vedi chmod).

NOME

passwd - installa o cambia la password associata alla login

SINTASSI

passwd [nome-utente]

FUNZIONE

Questo comando inserisce o cambia password. Se il comando e' dato senza argomenti cambia la password dell'utente che e' collegato, altrimenti cambia quella dell'utente specificato.

Se siamo nel caso di inserire una password, il comando ci chiede 2 volte (nel caso che la prima volta abbiamo sbagliato a digitare) la password senza che i caratteri da noi scritti ci appaiono su terminale (ci potrebbe essere qualche curiosone che guarda in quel momento). Solo se la stringa letta la prima volta corrisponde alla seconda, viene installata la password.

Nel caso di cambiare password, ci verra' chiesta prima

la vecchia password e solo nel caso che abbiamo dimostrato di conoscerla ci verra' chiesta la scrittura della nuova (sempre 2 volte).

La password dev'essere lunga almeno 4 caratteri o 6 caratteri a seconda che siano caratteri svariati o simili. In ogni caso, insistendo per 3 volte, viene accettata anche una password corta. Non ci sono limitazioni sulla lunghezza anche se vengono di fatto riconosciuti solo i primi 8 caratteri.

Solo l'utente stesso o un operatore puo' cambiare una password; l'utente deve sempre dimostrare di conoscere la precedente.

Il secondo campo (separato da un :) del file /etc/passwd, che e' normalmente leggibile a tutti, contiene le password di tutti gli utenti scritte secondo un codice interno che varia di caso in caso.

3.8. DOCUMENTAZIONE IN LINEA : MAN

Il comando man serve per reperire documentazione sui comandi di sistema. La struttura di essi e' simile a quella della descrizione dei comandi in questo testo.

NOME

man - stampa documentazione di un comando

SINTASSI

man [opzione ...] [capitolo] nome-comando

FUNZIONE

Man visualizza il manuale del comando specificato nel capitolo specificato. Se non e' stato specificato il capitolo, viene stampato l'intero manuale. Senza opzioni visualizza un manuale gia' impaginato.

-t Impagina tramite il comando troff.

-n Impagina tramite il comando nroff.

-c Significa -nc, filtra l'output attraverso il programma more.

-k Visualizza l'output su un terminale Tektronix

usando i programmi troff e tc.

-w Stampa solamente i path name delle sezioni del manuale.

ESEMPI

1) man -w chmod

Visualizza

/usr/man/man1/chmod.1.C /usr/man/man2/chmod.2.C

i path name delle (2 in questo caso) sezioni del manuale di chmod.

2) man -c 2 chmod

Visualizza la seconda sezione del manuale chmod facendolo impaginare da nroff e filtrare da more.

IMPERFEZIONI

C'e' l'inconveniente che quando il manuale viene ridiretto su un file, solitamente contiene i caratteri che servono per le sottolineature su stampante. Nell'editor vi si vedranno questi i due caratteri "_^H" prima di ogni carattere che deve essere sottolineato.

3.9. CHMOD

NOME

chmod - cambia le modalita' di accessi ai files

SINTASSI

chmod modo file ...

FUNZIONE

Il modo di ciascun file puo' essere assoluto o simbolico.

Il modo assoluto e' un numero ottale costruito con l'operazione OR dai seguenti modi:

4000 Attiva il bit s, detto "user ID on execution"

2000 Attiva il bit s per il gruppo

1000 Attiva il bit t, salva l'immagine del testo in

memoria dopo l'esecuzione per risparmiare tempo ad ogni successiva esecuzione. Questa modalita' e' permessa solo all'operatore.

0400 Permessi di lettura per il proprietario

0200 Scrittura per il proprietario

0100 Esecuzione per il proprietario

0070 Lettura, scrittura ed esecuzione per gli appartenenti al gruppo

0007 Lettura, scrittura ed esecuzione per tutti gli altri

Il modo simbolico ha la forma

[ugoa] [+-=] [rwxst]

u sta per l'utente (user), g per il gruppo, o per gli altri (other) ed a per tutti (all).
+ aggiunge i modi specificati da [rwxst], - li toglie ed = li mette.

Solo il proprietario del file o l'operatore puo' cambiare il modo.

ESEMPI

1) chmod 0700 file ...

produce -rwx----- ai files indicati.

2) chmod 4644 file ...

produce -rwsr--r-- ai files indicati.

3) chmod a=r file

Con una situazione iniziale qualsiasi, viene cambiata in

r--r--r--

4) chmod ug+x file

aggiunge il permesso in esecuzione per l'utente e il gruppo a cui appartiene.

3.10. MESSAGGI TRA UTENTI : WRITE, MAIL

NOME

write - scrive ad un altro utente

SINTASSI

```
write <nome-utente>
write to tty<numero-tty>
```

FUNZIONE

Write copia linea per linea il messaggio scritto sul terminale indicato. Dopo aver dato il comando, l'utente ricevente vede comparire il messaggio

```
Message from <nome-utente> <nome-terminale>
```

La comunicazione ha termine quando il mittente trasmette il carattere di end of file <ctrl>D. A questo punto il ricevente vedra' comparire

EOT

Se si vuole scrivere ad un utente che e' collegato su piu' di un terminale, bisogna specificare il numero del tty.

Il permesso di ricevere messaggi puo' essere disattivato con l'istruzione

```
mesg n
```

e ripristinato con

```
mesg y
```

Alcuni comandi come pr e nroff disabilitano temporaneamente i messaggi durante la loro esecuzione per fare in modo che non si sovrappongano i due output. Se arriva un messaggio mentre si sta utilizzando l'editor vi, e' possibile ripristinare la pagina dando il carattere <ctrl>L.

Se una linea inizia con il carattere "~!", write chiama la shell per eseguire il resto della linea come comando.

NOME

mail - spedisce o riceve posta tra gli utenti

SINTASSI

mail [-r] [-q] [-p] [-f nome-file]
mail <nome-utente> ...

FUNZIONE

Mail senza argomenti visualizza i messaggi ricevuti in ordine inverso con il criterio di ultima arrivata, prima stampata. Questo ordine viene invertito se viene usata l'opzione -r. I messaggi ricevuti vengono memorizzati in un file della home directory chiamato mbox. Se e' stata data l'opzione -p, la posta viene visualizzata senza dover dare input, altrimenti tra un messaggio e l'altro mail aspetta eventuali comandi da. I comandi interni a mail ed il loro significato sono i seguenti :

<return>

Ritorna al messaggio seguente

d Cancella il messaggio corrente e prosegue con gli altri

p Visualizza il messaggio

- Va al messaggio precedente

s file ...

Salva il messaggio nel file indicato (mbox per default)

w file ...

Salva il contenuto del messaggio senza intestazione, nel file indicato (mbox per default)

m [nome-utente]

Manda un mail all'utente specificato il cui contenuto e' quello del messaggio corrente.

<ctrl>D

Manda i messaggi non esaminati in mbox ed esce dal mail

q Come <ctrl>D

x Esce dal mail senza modificare mbox

!**<comando>**

Esegue <comando> in shell

? Stampa una sintesi dei comandi

Un interrupt (<delete> o altro) ferma la stampa della lettera corrente. L'opzione -q fa in modo che dopo un interrupt mail esca senza modificare mbox.

L'opzione -f fa in modo che il file indicato (per esempio, "mbox") sia preso come se fosse la nostra cassetta delle lettere (mailbox).

Quando viene spedita una lettera e' necessario dare uno o piu' nomi di utenti, scrivere il titolo della lettera sotto la richiesta

Subject:

il testo da spedire ed il carattere <ctrl>D.

Quando un utente si collega al sistema viene informato della presenza di posta con il messaggio "You have mail".

3.11. PARTI DI FILES: HEAD, TAIL

NOME

head - visualizza le prime linee dei files

SINTASSI

head [-numero] [file ...]

FUNZIONE

Head trasferisce in output le prime n righe dei files specificati o, se non sono stati indicati, lo standard input. Se e' stato omesso il numero di linee, per default viene assunto 10.

ESEMPI

1) head -50 file ... > inizi

Manda sul file inizi le ultime 50 linee dei files specificati.

NOME

`tail` - visualizza l'ultima parte di un file

SINTASSI

`tail +numero[lbc] [file]`

FUNZIONE

Tail trasferisce il file in output iniziando da un certo punto in poi. Se il nome del file viene omissso, viene assunto lo standard input.

Il trasferimento inizia dal punto +numero dall'inizio del file o -numero dalla fine del file. L'unita' di misura di numero viene espressa in linee, blocchi (512 caratteri) o caratteri a seconda rispettivamente delle opzioni l, b oppure c. Se non viene specificata l'unita' di misura, il conteggio e' per linee.

ESEMPI

1) `tail -15 prova`

Visualizza le prime 15 linee del file prova.

2) `tail +10b prova.`

Visualizza la parte del file prova che incomincia dal decimo blocco in poi.

3) E' possibile, usando una pipe, prendere una parte intermedia di un file

`tail +100 file | head -50`

Visualizza le 50 linee del file comprese tra la 101 e la 150.

3.12. SPEZZARE FILES : SPLIT**NOME**

`split` - spezza un file in parti

SINTASSI

`split [-n] [file [nome]]`

FUNZIONE

Split legge il file (lo standard input se omissso) e lo divide in tanti files piu' piccoli di lunghezza n

righe (1000 righe per ogni file se n e' stato omesso). Supponiamo di aver dato il nome "pezzo", a questo punto verranno creati tanti files di n righe chiamati pezzoaa, pezzoab, pezzoac ecc... Se nome viene omesso, viene assunto il nome x.

ESEMPI

1) split -50 file uscita

Spezza il file in tanti files di 50 linee ciascuno chiamati uscitaaa, uscitaab eccetera.

3.13. CONTEGGI SU FILES: WC

NOME

wc - conta parole

SINTASSI

```
wc [ -lwcpt ] [ -b<numero baud> ] [ -s<lunghezza-
pagina> ] [ -u ] [ -v ] [ file ... ]
```

FUNZIONE

Wc conta linee, parole e caratteri e opzionalmente pagine e il tempo di trasferimento su terminale di uno o piu' files. Una parola e' definita come una sequenza di caratteri delimitata da spazi o tab o <return>.

Le opzioni "lwcpt" stanno per il conteggio rispettivamente di linee, parole, caratteri, pagine (si assumono 66 righe per pagina) e tempo. Per default vengono assunte le opzioni -lwc.

-b<numero-baud>

Tiene conto della velocita' di trasmissione del terminale nel conteggio del tempo. Se non viene specificata tramite questa opzione, la velocita' viene assunta in 300 baud.

-s<lunghezza-pagina>

Specifica la lunghezza della pagina al posto di 66 righe.

-u

Informa che il tempo dev'essere basato in tempo di trasmissione uucp (reti che collegano macchine che adottano il sistema operativo UNIX), circa il 90% della velocita' normale.

-v Aggiunge le opzioni -pt ed intesta l'output con
 lines words chars pages time@300

ESEMPI

1) wc -v -b9600 -s15 testo

Viene richiesto ogni tipo di conteggio e si specifica che la velocita' di trasmissione e' di 9600 baud e che le pagine sono lunghe 15 linee. Ci produce qualcosa come

lines	words	chars	pages	time@9600
1466	7694	44080	98	45.0 se testo

IMPERFEZIONI

Il tempo calcolato e' puramente indicativo

3.14. INTERVALLI DI TEMPO: SLEEP, AT

NOME

sleep - sospende l'esecuzione per un determinato numero di secondi.

SINTASSI

sleep <numero-secondi>

FUNZIONE

Sleep sospende l'esecuzione per un numero di secondi che varia da 1 a 65536. E' usato per eseguire un comando dopo un certo periodo di tempo o per eseguire spesso uno o piu' comandi.

ESEMPI

1) sleep 600 ; comando &

Esegue il comando dato dopo 600 secondi (10 minuti). Mentre lo sleep e' in funzione, possiamo proseguire il lavoro poiche' il comando e' stato mandato in background con la &.

2) while <condizione>

```

do
    <comando>
    sleep 20
done

```

Esegue il <comando> ogni 20 secondi finche' e' verificata <condizione>.

NOME

at - esegue comandi alla data specificata

SINTASSI

at orario [giorno] [file]

FUNZIONE

File dev'essere un file di comandi shell. At si tiene una copia del file nominato nella directory /usr/spool/at facendola eseguire dalla shell al tempo specificato. All'inizio dei comandi viene immesso il comando "cd" alla directory corrente seguito dagli assegnamenti delle variabili della nostra shell.

Orario puo' essere tra 1 e 4 caratteri. I primi due sono le ore e gli ultimi due i minuti. Puo' essere accodato "am" o "pm" per specificare prima o dopo mezzogiorno.

Giorno puo' essere (1) un giorno del mese seguito dal numero del giorno o (2) un giorno della settimana; se e' seguito dalla parola "week", viene capito come "della settimana prossima".

Lo standard output e lo standard error, se non e' stato ridiretto, viene perso.

ESEMPI

1) at 1830 apr 12 file

Esegue il file di comandi alle 18.30 del 12 aprile

2) at 10pm fr week file > out

Esegue il file alle 10 di sera di venerdi' prossimo, ridirigendo l'output su "out".

3.15. PROCESSI E LORO FINE: WAIT, PS, KILL

NOME

wait - aspetta la fine dei processi in background

SINTASSI

wait

FUNZIONE

Wait sospende l'esecuzione finche' non sono finiti tutti i processi iniziati con una &.

Se wait riceve un interrupt (<delete>), riporta i comandi ancora attivi in background con il relativo numero di processo. Questo puo' essere molto utile per (1) conoscere quanti e quali nostri processi sono ancora attivi (2) fare terminare un processo con l'istruzione kill se non ci ricordiamo il numero ad esso associato.

NOME

ps - situazione dei processi

SINTASSI

ps [aglwx] [t <nome-tty>]

FUNZIONE

Ps stampa informazioni sui processi attivi. Senza opzioni, ps stampa lo stato dei nostri processi con informazioni sul numero del processo, il terminale in cui e' in esecuzione, il tempo di cpu usato fino a quel momento ed il nome del comando. Le seguenti opzioni possono essere combinate tra di loro :

- a Da' informazioni su tutti i processi, al posto dei soli processi di cui siamo proprietari.
- g Informazioni su tutti i processi. Normalmente non vengono segnalati i processi il cui nome inizia con un "~", come le shell (-sh) e i - <carattere> (terminale libero in attesa di un

collegamento).

- 1 Riporta un elenco esteso dei processi il cui significato e' descritto dopo queste opzioni.

t<nome-tty>

Restringe l'analisi dei processi a quelli del terminale indicato. ttty3 per il terminale numero 3, tconsole per la console. Questa opzione deve essere l'ultima tra le date.

- x Riporta anche i processi che non sono eseguiti su nessun terminale. A questa categoria appartengono i processi di sistema come lo swapper che gestisce i trasferimenti tra memoria centrale e disco e viceversa.

I campi di intestazione dell'elenco esteso hanno il seguente significato :

- F Flag associato al processo. Essi sono definiti nelle linee #define in /usr/include/sys/proc.h

- S Definisce lo stato del processo.

0 : non esistente
S : sleeping
W : waiting (in attesa)
R : running (in esecuzione)
I : in uno stato intermedio
Z : terminato
T : interrotto

- UID Il numero della user-id del proprietario del processo. Guardare nel file /etc/passwd per conoscere il nome della user-id. Il numero rappresenta il terzo campo di ogni riga del file.

- PID Il numero del processo.

- PPID Il numero del processo padre. Ogni comando eseguito e' figlio della shell, se abbiamo dato un comando all'interno di more o di qualche altro programma, esso sara' figlio di more o di chi altro. Il padre della shell con cui ci colleghiamo e' sempre il processo /etc/init (con numero 1).

- CPU L'utilizzo del processore per la schedulazione.

PRI La priorita' del processo, numeri alti significano una bassa priorita'.

NICE Va da 0 a 39 e rappresenta un valore utilizzato nel calcolo delle priorita'. Nice 0 e' la massima priorita'.

ADDR L'indirizzo di memoria se il processo e' residente nella stessa, altrimenti l'indirizzo di disco.

SZ Il numero di blocchi che occupa il processo in memoria.

WCHAN

L'evento per cui il processo e' in waiting o in sleeping; se e' bianco, il processo e' in esecuzione.

TTY Il terminale a cui e' associato il processo. I processi swapper ed /etc/init hanno un ?.

TIME Il tempo di cpu usato.

COMMAND

Il comando con il suo argomento.

IMPERFEZIONI

La situazione dei processi puo' cambiare mentre ps e' in esecuzione. Non sempre e' presente il campo COMMAND.

NOME

kill - termina un processo

SINTASSI

kill [-<numero-segnale>] numero-processo ...

FUNZIONE

Kill manda il segnale 15 (terminazione) ai processi specificati. Se viene dato il numero del segnale preceduto da un -, viene trasmesso il segnale stesso al posto del 15. Se kill non riesce a far terminare il processo, "kill -9 <processo>" e' un segnale che non puo' essere ignorato e riesce certamente nel suo scopo.

"kill -9 0" termina tutti i nostri processi compresa la shell.

E' possibile terminare solo i nostri processi, fatta eccezione per l'operatore.

Per terminare processi in background di cui non ci ricordiamo il numero, si puo' seguire 4 vie :

- 1) Visualizziamo il contenuto della `#!` oppure `$child` (a seconda che la shell sia `/bin/sh` oppure `/bin/csh`) che contengono il numero dell'ultimo processo mandato in background, nel caso che ci interessi solo l'ultimo.
- 2) Nel caso che il processo sia figlio diretto della nostra shell (un processo in background potrebbe essere stato dato all'interno di un'altro comando o di un file di comandi shell, in tal caso non viene raggiunto dalla wait), si esegue una wait e poi si trasmette un interrupt per avere l'elenco dei processi mandati in background.
- 3) Si esegue `ps` e si ricava il numero.
- 4) Caso estremo, si da' `"kill -9 0"` e ci si ricollega al sistema.

3.16. ORDINAMENTO DI FILES: SORT, TSORT

Il comando `sort` serve per ordinare files. E' un comando piuttosto potente se usato correttamente. Per apprendere l'uso avanzato di `sort`, e' necessaria una particolare attenzione.

NOME

`sort` - ordina files o fa il merge di files (ordina piu' files tra di loro)

SINTASSI

`sort [-mubdfinrtx] [+pos1 [-pos2]] ... [-o name] [-T directory] [name] files`

FUNZIONE

`Sort` ordina per ordine alfabetico (in ordine di caratteri ASCII, per cui le cifre vengono prima delle maiuscole che vengono prima delle minuscole) le linee di tutti i files dati insieme, e scrive il risultato

sullo standard output. E' possibile ordinare un certo campo di un file invece di ordinare l'intera linea. Le seguenti opzioni possono apparire singolarmente o combinate tra di loro.

b Ignora i caratteri blank e i tabs nella comparazione dei campi

d Ordina solo i caratteri alfanumerici. Tutti i caratteri che non non siano lettere, cifre o spazi bianchi, non sono significativi nei confronti.

f Considera i caratteri maiuscoli come minuscoli.

n Ordina in ordine crescente una stringa iniziale numerica, che puo' contenere il segno e decimali dopo il punto. Questa opzione implica l'opzione b.

r Inverte il senso di ordinamento. Usare entrambe le opzioni nr per avere un ordinamento decrescente numerico.

i Nelle comparazioni non numeriche ignora i caratteri che nella sequenza ASCII escono dai limiti 32-126.

tc Assume il carattere c come separatore di campi. Se non e' altrimenti specificato, la chiave da ordinare e' rappresentata dall'intera linea. La notazione +pos1-pos2 restringe la chiave da ordinare ad uno specifico campo della linea che incomincia a pos1 e finisce appena prima di pos2. Entrambe pos1 e pos2 hanno la forma m.n, seguiti opzionalmente da uno o piu' delle opzioni precedenti, dove m indica il numero di campi da saltare dall'inizio della linea ed n il numero di caratteri da saltare ulteriormente. Se manca .n, viene assunto .0, se manca -pos2, viene assunta la fine della linea. Se e' presente l'opzione t, i campi sono stringhe separate dal carattere indicato, altrimenti sono stringhe separate da blanks. Quando ci sono piu' chiavi da ordinare, i campi successivi vengono ordinati solo dopo che quelli precedenti sono risultati uguali.

c Verifica se un file e' gia' ordinato nella maniera indicata. Se lo e', il comando non da output, altrimenti evidenzia qual'e' il punto in cui non e' piu' stato rispettato l'ordinamento.

m Unisce in ordine files precedentemente ordinati (merge).

o L'argomento successivo e' il nome del file su cui deve andare l'output.

T L'argomento successivo e' il nome di una directory in cui vengono creati i file temporanei. Normalmente vanno in /usr/tmp e in /tmp

u Sopprime in output, a parte una, tutte le linee uguali tra loro.

ESEMPI

1) sort -uf file1 file2 ..

Ordina i file indicati dando in output una sola tra piu' linee uguali, non distinguendo tra maiuscole e minuscole

2) sort -t: +2nr /etc/passwd

Il separatore tra i campi e' :, stampa il file delle password ordinato secondo il campo del numero delle user-id (terzo campo separato da :) in ordine decrescente.

3) wc -l * | sort -n

Ordina per numero di righe i files della current directory.

4) wc | sort -n +1

Ordina secondo il numero di parole (secondo campo) i files della current directory. Se non e' stato specificato il separatore tra campi, viene assunto lo spazio o tab come separatore, per cui +1 indica di considerare il campo successivo dopo lo spazio.

5) sort -cfr file

Verifica che il file sia ordinato in maniera inversa senza distinzione tra maiuscole e minuscole.

IMPERFEZIONI

Le linee piu' lunghe di 80 caratteri sono troncate.

NOME

tsort - ordinamento topologico

SINTASSI

tsort [file]

FUNZIONE

Tsort produce in output una lista ordinata di item (stringhe non vuote) in accordo con il parziale ordinamento degli item nel file di input. Se il file e' omesso, viene preso lo standard input.

L'input consiste di coppie di item separate da uno spazio. Coppie di item differenti indicano ordinamento, coppie di item identici indicano presenza ma non ordinamento.

ESEMPI

1) tsort

Dando in input i seguenti item

```
marco luigi
clara marco
stefano luigi
sara clara
edo clara
```

e' stato comunicato che marco viene prima di luigi, clara prima di marco ecc... Sono state cioe' definite le grandezze. Chiudiamo l'input con <ctrl>D ed otteniamo

```
stefano
sara
edo
clara
marco
luigi
```

in accordo con le informazioni ricevute, l'ordinamento risulta essere questo. Ulteriori specifiche otterrebbero un'altro ordinamento.

3.17. OCCUPAZIONE DI MEMORIA: DU, DF, QUOT, COMPACT**NOME**

du - riporta l'occupazione su disco

SINTASSI

```
du [ -s ] [ -a ] [ file ... ]
```

FUNZIONE

Du riporta il numero di blocchi occupati in tutte le sottodirectory alle directory specificate. Senza argomenti, viene assunta la current directory.

- s Riporta solamente il totale dei blocchi senza analizzare le directory interne.
- a Riporta l'occupazione di ogni file incontrato durante il cammino

Un file con due links viene contato una volta sola.

ESEMPI

1) du -a

Riporta l'occupazione in blocchi di ogni file della directory corrente e di quelle ad essa interne.

NOME

df - occupazione di disco

SINTASSI

df [<nome-filesystem> ...]

FUNZIONE

Df riporta lo spazio occupato su disco dai vari filesystem presenti (il cui numero varia a seconda di come e' stata configurato il sistema). Se specificati, riporta informazioni limitate ad essi.

ESEMPI

1) df

Produce

Filesystem	Mounted on	blocks	used	free	% used
/dev/rp1	/	15358	13765	1593	90%
/dev/rp3	/usr/1	14470	12027	2443	83%

/dev/rp1 e /dev/rp3 sono due file speciali che corrispondono a parti del disco. Questo e' un sistema con due filesystem : /dev/rp1 contiene tutti i files di sistema e /dev/rp3 la parte riservata agli utenti. Vengono riportati il numero di blocchi assegnati al

filesystem, i blocchi usati, quelli ancora liberi e la percentuale di quelli usati rispetto a quelli assegnati.

NOME

quot - occupazione del file system per ogni utente.

SINTASSI

quot [opzione] ... [filesystem]

FUNZIONE

Quot visualizza, rispetto al filesystem indicato, in ordine di grandezza decrescente, il numero di blocchi occupati da ogni utente. Se non e' stato specificato il file system, ne viene assunto uno di default.

-c Stampa tre colonne di cui la prima indica la grandezza in blocchi, la seconda il numero di files di quella grandezza e la terza la somma cumulativa dei blocchi dei files di quella dimensione o piu' piccoli.

-f Stampa il numero di blocchi in ordine decrescente e il numero di files posseduti da ogni utente.

NOME

compact, uncompact, ccat - compatta, scompatta e visualizza file compattati.

SINTASSI

compact [file ...]
uncompact [file ...]
ccat [file]

FUNZIONE

Compact riduce la dimensione di un file. Il file viene trasformato in un codice e viene aggiunto il suffisso ".C" al nome del file stesso. Lo scopo di cio' e' ridurre l'occupazione di memoria di massa.

Se vengono omessi i file, viene compattato lo standard input sullo standard output.

I primi due byte di un file compattato segnalano che il file e' di quel tipo, per proibire un ulteriore

compattamento. Se il file e' collegato ad altri tramite links, il legame viene tolto.

Dopo aver compactato un file, compact segnala la percentuale di riduzione. I valori medi sono :

38%	per testi
43%	per programmi Pascal
36%	per programmi C
19%	per codici binari

Uncompact e' l'opposto di compact: espande un file compactato ripristinando le informazioni originarie e toglie il suffisso ".C" dal nome.

Ccat permette di visualizzare un file compactato senza usare l'istruzione uncompact.

ESEMPI

1) Alla fine del lavoro, prima di scollegarsi dal sistema, si puo'dare

```
compact * 2> /dev/null &
```

per ridurre l'occupazione di memoria della directory. Viene ridiretto l'output e l'error sul file di "scarico" /dev/null per evitare l'interferenza del comando compact con il terminale.

2) ccat file | pr -3 | lpr

Manda su stampante il file su tre colonne facendolo rimanere nella forma compactata.

IMPERFEZIONI

Il nome di un file da compactare non puo' essere piu' lungo di 12 caratteri (l'ultima parte del path name).

3.18. COMPARAZIONE FILES: CMP, DIFF

NOME

cmp - compara due files

SINTASSI

```
cmp [-l ] [-s ] file1 file2
```

FUNZIONE

Confronta file1 e file2. Senza opzioni, cmp non riporta messaggi in output se i files sono identici; altrimenti riporta il numero del byte e il numero della linea in cui e' stata trovata la prima differenza.

- l Visualizza per ogni differenza incontrata, il numero del byte in decimale corrispondente alla posizione del file e due numeri ottali che corrispondono ai caratteri ASCII dei due files.
- s Non visualizza differenze, al termine, modifica il valore della variabile contenente lo status. Se tale valore e' 0, i files sono identici; 1 per files differenti e 2 per files che non esistono o per mancanza di argomenti.

NOME

diff - differenzia files e compara le directory

SINTASSI

```
diff [-l] [-r] [-s] [-S<file>] [-cefh] [-b] dir1 dir2
diff [-cefh] [-b ] file1 file2
diff [-b ] file1 file2
```

FUNZIONE

Se entrambi gli argomenti sono directory, diff ordina i contenuti delle stesse per nome e li compara tra di loro. Vengono segnalati i files che compaiono solo nella prima o nella seconda directory e i files comuni ad entrambe. Le opzioni sono le seguenti:

- l Causa un output esteso. Vengono riassunti i files appartenenti solo ad una delle due directory e quelli in comune.
- s Visualizza le differenze dei files con lo stesso nome.
- r Viene applicato diff ricorsivamente alle sottodirectory comuni.
- S<file>
Comincia il confronto dal file indicato.

Quando gli argomenti sono files normali, e quando compara file di tipo testo che differiscono durante il confronto di directory, diff dice quali linee devono essere cambiate nei files per portarli identici tra di loro. Se file1 e' una directory, allora viene confrontato con file2 il file della directory file1 che ha lo stesso nome di file2 (e viceversa).

Il formato di default dell'output contiene linee della forma

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

Queste linee assomigliano a comandi dell'editor ed per convertire file1 in file2. I numeri dopo le lettere (a,d,c) si riferiscono a file2.

Ogni linea di quel formato e' seguita dalla lista delle linee che si riferiscono al primo file se precedute da un "<" e al secondo se precedute da un ">".

Eccetto che per l'opzione -b che puo' essere data insieme alle altre, le seguenti opzioni sono in conflitto tra di loro.

- e Produce un testo di comandi per l'editor ed che ricrea file2 da file1. Vengono aggiunti comandi in shell quando vengono comparate directory con l'opzione -e in modo che il risultato sia un file di comandi shell per convertire file di tipo testo che sono comuni nelle due directory dallo stato in dir1 a quello in dir2.
- f Produce un testo simile a quello di -e in ordine opposto e non adatto per ed.
- c Produce le differenze elencando le linee opportune. L'output inizia con l'identificazione dei files e la loro data di creazione, successivamente ogni cambiamento e' separato da una linea con una dozzina di *. Le linee rimosse da file1 sono contrassegnate ad inizio linea con "-"; quelle aggiunte a file2 con "+"; linee che sono cambiate da un file all'altro con "!".
- h E' piu' efficiente. Funziona solo se i cambiamenti sono piccoli e ben distinti.
- b Ignora gli spazi e i tabs e altre stringhe di spazi vengono considerate uguali.

Lo stato di uscita e' 0 se non ci sono differenze,
1 per differenze e 2 per comando incorretto.

3.19. RICERCA DI PATTERNS: GREP, LOOK

Pattern e' un'espressione formata da una sequenza di caratteri che identifica una o piu' parole da ricercare.

NOME

grep, egrep, fgrep - ricerca un pattern in un file

SINTASSI

```
grep [ opzioni ] espressione [ file ] ...
egrep [ opzioni ] [ espressione ] [ file ] ...
fgrep [ opzioni ] [ stringhe ] [ file ] ...
```

FUNZIONE

I comandi della famiglia di grep cercano nei file di input (lo standard input per default) linee che verificano la condizione specificata nel pattern. Normalmente ogni linea trovata e' copiata in output; se non e' usata l'opzione -h, viene visualizzato il nome del file (nel caso che la ricerca sia su piu' di un file).

I patterns di grep sono espressioni nello stile di ed; i patterns di egrep sono vere e proprie espressioni regolari; i patterns di fgrep sono stringhe fisse, ha una ricerca molto veloce. Grep ed egrep riconoscono propri metacaratteri.

- v Visualizza le linee che NON contengono il pattern.
- c Riporta solo il conteggio delle linee soddisfacenti l'espressione.
- l Riporta solo i nomi dei files che contengono il pattern.
- n Ogni linea e' preceduta dal proprio numero di linea.
- b Ogni linea e' preceduta dal numero del blocco in cui e' stata trovata.

- s Senza output, viene modificato solo la variabile di status.
- h Non visualizza i nomi dei files.
- y Nella ricerca non distingue tra maiuscole e minuscole (solo grep).
- e<espressione>
Si differenzia dall'espressione normale dal fatto che e' utile quando l'espressione inizia con "-".
- f<file>
L'espressione per egrep e l'elenco di stringhe per fgrep sono prese dal <file>.
- x Vengono visualizzate le linee che contengono solamente le stringhe (solo fgrep).

Quando vengono usati metacaratteri, e' opportuno racchiudere il pattern tra apici poiche' gli stessi hanno significato anche per la shell.

Significato dei metacaratteri in grep ed egrep:

Un \ seguito da un singolo carattere seleziona il carattere.

I caratteri ^ e \$ si riferiscono rispettivamente all'inizio e alla fine della linea.

Un . (punto) ha il significato di "qualsiasi carattere".

Un * ha il significato di "qualsiasi numero di ripetizioni, anche zero".

Una stringa racchiusa tra parentesi quadre seleziona qualsiasi carattere all'interno delle stesse. Caratteri consecutivi possono essere abbreviati con un _ meno come in "[a-z0-9]". Una] all'interno delle quadre puo' comparire solo come primo carattere della stringa. Il carattere "-" deve essere piazzato in un posto in cui non si confonda come separatore del range.

Altri metacaratteri di egrep:

Una espressione seguita da *, +, ?, seleziona rispettivamente una sequenza di almeno zero, almeno una, zero o una ricorrenza dell'espressione.

Due espressioni separate da | o da una nuova linea

incontrano sia la prima che la seconda espressione (operatore or).

Un'espressione racchiusa tra parentesi seleziona una selezione per l'espressione.

Le precedenze degli operatori nello stesso livello di parentesi e' [], * + ?, | e nuova linea.

Se sono state trovate ricorrenze lo status di uscita e' uguale a 0, 1 per nessuna ricorrenza e 2 per errori nella sintassi del comando.

ESEMPI

1) `grep '[^M-Z]' file(s)`

Trova tutte le linee che iniziano (carattere ^) col il insieme di caratteri che va da M a Z.

2) `grep 'c.t' file(s)`

Trova le linee che contengono una c seguita da qualsiasi altro carattere (il carattere .) e da una t.

Vengono allora selezionate linee contenenti

```
cat
il carattere c trova
citta'
stracotto
```

3) `grep -n 'te.*lo' file(s)`

Visualizza in output le linee (con il proprio numero per l'opzione -n) contenenti "te" seguito da qualsiasi numero, anche zero, (il carattere *) di qualsiasi carattere e da "lo".

Verranno selezionate linee del tipo

```
corrente, vediamo con
fisicamente lo stesso
permette solo
interpretarlo
```

4) `egrep '^5+' file(s)`

Vengono incontrate le linee che iniziano (carattere ^) con uno o piu' 5 (carattere +).

In un file tipo

```

34
345
534
552
51
234
253

```

vengono selezionate le linee

```

534
552
51

```

5) `egrep '^alo$' file(s)`

Vengono selezionate le linee che iniziano per a e (carattere !) quelle che finiscono per o.

6) `egrep -v '[A-Z]![a-z]' file(s)`

Seleziona le linee che non contengono (opzione -v) caratteri alfabetici maiuscoli o minuscoli.

7) `fgrep 'uno
due
tre' file(s)`

Questo comando, diviso in piu' linee, trova le tre stringhe "uno" "due" "tre".

NOME

`look` - trova linee in files ordinati

SINTASSI

`look - [-df] stringa [file]`

FUNZIONE

`Look` legge un file ordinato e stampa le linee che iniziano con la stringa. Usa una ricerca binaria, per cui e' piuttosto veloce.

`-d` Ordine alfabetico, solo lettere, numeri, tabs e spazi partecipano nelle comparazioni.

`-f` Le lettere maiuscole sono considerate uguali alle

minuscole.

Se il file non viene specificato, viene assunto il file /usr/dict/words con le opzioni -df.

3.20. CONDIZIONI: TEST

NOME

test - comando di condizioni

SINTASSI

test <espressione>

FUNZIONE

Test valuta l'espressione, se il suo valore e' vero ritorna zero come stato di uscita altrimenti ritorna uno stato diverso da zero. Ritorna inoltre non zero se non ci sono argomenti.

-r <file>

Vero se il file esiste ed e' leggibile

-w <file>

Vero se il file esiste ed e' scrivibile

-f <file>

Vero se il file esiste e non e' una directory

-d <file>

Vero se il file esiste ed e' una directory

-s <file>

Vero se il file esiste ed ha una grandezza piu' grossa di zero

-t [fildes]

Vero se il file il cui numero del file descriptor e' fildes (1 per default) e' associato ad un terminale

-z s1

Vero se la lunghezza della stringa s1 e' zero.

-n s1

Vero se la lunghezza della stringa s1 non e' zero.

```

s1 = s2
    Vero se le stringhe s1 e s2 sono uguali

s1 != s2
    Vero se le stringhe s1 e s2 non sono uguali

s1      Vero se s1 non e' una stringa nulla

n1 -eq n2
    Vero se gli interi n1 ed n2 sono algebricamente
    uguali. Altre comparazioni : -ne ( diverso ); -gt
    ( maggiore ); -ge ( maggiore o uguale ); -lt
    ( minore ); -le ( minore o uguale ).

```

Queste primitive possono essere combinate tra loro con i seguenti operatori:

```

!    operatore not
-a   operatore and
-o   operatore or
( espressione ) parentesi per i gruppi

```

-a ha precedenza rispetto a -o. Le parentesi debbono essere precedute da \.

ESEMPI

Test e' usato soprattutto in file di comandi shell. Nell'esempio seguente, viene controllato se il primo argomento nel richiamo di un file di comandi e' un file esistente e non e' una directory.

```

if test -f $1
then
< serie di istruzioni >
else echo $1 ': non e' stato trovato'
fi

```

3.21. GESTIONE DEL VIDEO TERMINALE: STTY, RESET

NOME

stty - assegna i parametri del terminale

SINTASSI

stty [parametro ...]

FUNZIONE

Stty assegna alcuni parametri di I/O per il video terminale. Senza argomenti, riporta i parametri assegnati fino a quel momento. I parametri sono i seguenti: (una c significa <carattere>)

erase c

Assegna a c il carattere di erase (annulla il carattere precedentemente digitato). Per default e' "#".

kill c

Assegna a c il carattere di kill (annulla i caratteri digitati della riga corrente). Per default e' "@".

intr c

Assegna a c il carattere di interrupt (interrompe un processo in esecuzione). Per default e' <delete>.

quit c

Assegna a c il carattere di quit (interrompe un processo creando un file chiamato core contenente l'immagine in memoria del processo). Per default e' <ctrl>\.

stop c

Assegna a c il carattere di stop (sospende momentaneamente l'output su terminale). Per default e' <ctrl>S.

start c

Assegna a c il carattere di start (riprende l'output interrotto dal carattere di stop). Per default e' <ctrl>Q.

eof c

Assegna a c il carattere di end of file (comunica al sistema la fine dell'input). Per default e' <ctrl>D.

ex

Assegna i caratteri di erase e kill ai normali # e @.

raw

Non vengono piu' riconosciuti i caratteri di erase, kill, interrupt, tab, quit, end of line ed end of file.

-raw

Disabilita il raw. Deve essere trasmesso con <line feed> o con <ctrl>J.

cooked
Come -raw

cbreak
Legge un carattere alla volta. Ogni carattere trasmesso viene letto senza aspettare <return>. Non vengono riconosciuti i caratteri di erase e kill.

-cbreak
Fa in modo che i caratteri siano letti al momento del <return>.

nl Accetta solo <line feed> per terminare una linea.

-nl Abilita <return> per terminare una linea e trasmette CR-LF sia per <return> che per <line feed>.

-echo
Non visualizza i caratteri digitati.

echo
Visualizza i caratteri digitati

lcase
Trasforma i caratteri minuscoli in maiuscoli.

-lcase
Non trasforma caratteri.

-tabs
Sostituisce i tabs con spazi.

tabs
Mantiene i tabs.

50 75 110 134 150 200 300 600 1200
1800 2400 4800 9600 exta extb
Assegna, se possibile, la velocita' di trasmissione del terminale in baud.

everything
Riporta lo stato di tutte le opzioni del terminale.

NOME

reset - riporta i parametri del terminale a' valori

standard

SINTASSI

reset

FUNZIONE

- Reset riporta i bit del terminale a valori standard, con <ctrl>H come carattere di erase e @ come carattere di kill. Reset e' molto utile quando vogliamo uscire dalla modalita' raw. In tal caso reset va trasmesso con <line feed>.

3.22. CONTENUTO DI FILES: SEE, STRINGS, FILE

Non sempre e' possibile con i comandi vi, cat, more, ecc..., visualizzare tutti i caratteri di un file, determinarne il tipo o il contenuto. Raggiungono lo scopo i seguenti programmi:

NOME

see - visualizza il contenuto di uno o piu' files

SINTASSI

see [file ...]

FUNZIONE

See visualizza in maniera leggibile un file che contiene caratteri non visualizzabili. I caratteri di controllo sono stampati come ^I per i tab. Delete e' stampato come ^?.

NOME

strings - trova le stringhe stampabili in un file di tipo binario

SINTASSI

strings [-] [-o] [-numero] file ...

FUNZIONE

Strings cerca stringhe di tipo ASCII in un file binario. Una stringa viene intesa come una sequenza di 4 o piu' caratteri visualizzabili terminati con newline

o null. Se e' stata data l'opzione -o, ogni stringa e' preceduta dalla propria posizione all'interno del file (in ottale). Se e' stata data l'opzione -numero, quel numero viene utilizzato per determinare la lunghezza minima di una stringa, al posto di 4.

Strings e' utile per identificare file di tipo oggetto (formato obj, a.out) e per parecchie altre cose.

NOME

file - determina il tipo del file

SINTASSI

file file ...

FUNZIONE

File classifica ogni file secondo un tipo. Se un file e' di tipo ASCII, file legge i primi 512 byte per capire caratteristiche piu' specifiche. Classici tipi sono:

```

directory
roff, nroff, or eqn input text
ascii text
C program
English text
data
directory
commands text
```

IMPERFEZIONI

Files di comandi shell vengono spesso classificati come C program, files compattati come data.

3.23. RIPETIZIONI DI LINEE: UNIQ

NOME

uniq - riporta linee ripetute di un file

SINTASSI

uniq [-udc [+n] [-n]] [input [output]]

FUNZIONE

Uniq legge il file di input comparando linee adiacenti. Normalmente la seconda e le successive copie di una linea non vengono trasferite in output. Per default vengono assunte le opzioni -u e -d.

- u Non viene copiata in output neanche la prima delle linee contigue uguali.
- d Vengono visualizzate solo le linee che si ripetono, in una copia sola.
- c Genera un output in cui ogni linea e' preceduta dal numero di volte che e' ripetuta.
- n Vengono ignorati i primi n campi.
- +n Vengono ignorati i primi n caratteri.

3.24. REPERIMENTO FILES: FIND

NOME

find - trova files che rispondono a certe caratteristiche

SINTASSI

find lista-di-path-name espressione

FUNZIONE

Find discende ricorsivamente la gerarchia delle directory per ogni path name della lista cercando i files che soddisfano una certa condizione. Nelle descrizioni che seguono, l'argomento n e' un intero decimale che puo' essere usato con +n (significa maggiore di n), -n (minore di n) oppure n (esattamente n).

-name <nomefile>

Vero se <nomefile> incontra il filename corrente nel percorso del find. Possono essere usati i metacaratteri della shell, in tal caso e' consigliabile racchiudere il nome tra apici. Se si vuole cercare da /usr in poi tutti i nomi dei files che finiscono in .C, il comando sara' cosi' strutturato:

```
find /usr -name '*.C' .....
```

- user <nome-utente>
Vero se il file e' di proprieta' dell'utente
- group <nome-gruppo>
Vero se il file e' di proprieta' del gruppo.
- perm <numero-ottale>
Vero se i permessi del file corrispondono al numero ottale (vedi chmod)
- type <carattere>
<carattere> puo' essere b, c, d oppure f. Vero se il tipo del file e' b per file di tipo block special, c per character special, d per directory ed f per file normale.
- links n
Vero se il file ha n links.
- size n
Vero se il file occupa n blocchi di memoria (1 blocco e' 512 caratteri)
- inum n
Vero se il file ha i-number n
- atime n
Vero se il file non ha avuto alcun accesso da n giorni
- mtime n
Vero se il file non e' stato modificato da n giorni
- exec <comando>
Vero se l'esecuzione del comando ritorna 0 come stato di uscita. Alla fine del comando bisogna dare un ; preceduto da backslash (\). L'argomento {} e' rimpiazzato dal find con il pathname corrente della ricerca.
- ok <comando>
Come -exec con la differenza che il comando, per ogni file trovato, viene scritto in output e viene letta la conferma da input. Il comando viene eseguito solo se la risposta e' y.
- print
Sempre vero, visualizza il path name corrente.

-newer file

Vero se il file corrente e' stato modificato piu' recentemente del file dato come argomento.

Le opzioni possono essere combinate usando i seguenti operatori:

- 1) Un gruppo delimitato da parentesi contenente opzioni ed operatori. Le parentesi devono essere precedute da \.
- 2) L'operatore not ("!" per negare un'opzione).
- 3) Concatenazione di opzioni (l'operatore and e' implementato dalla vicinanza di due opzioni).
- 4) L'operatore or (-o).

ESEMPI

1) Si vogliono trovare in tutti i files della home directory (e di tutte le directory ad essa interne) tutti i file che sono piu' grossi di 4 blocchi, a quel punto compattarli:

```
find $HOME -size +4 -exec compact {} \;
```

2) A partire dalla directory /usr/utenti si vogliono trovare tutti i files di nostra proprieta':

```
find /usr/utenti -user $USER -print
```

3) Si vuole rimuovere in tutto il sistema tutti i files chiamati "a.out" o "*.o" che non hanno avuto alcun accesso per piu' di una settimana:

```
find / \(\ -name a.out -o -name '*.o' \) -atime +7 -exec  
rm {} \;
```

3.25. PRIORITA': NICE

NOME

nice, nohup - esegue un comando a bassa priorita'

SINTASSI

```
nice [ +-numero ] [ <comando> ]
nohup [ <comando> ]
```

FUNZIONE

Nice esegue il comando con bassa priorita'. Se l'argomento numero e' presente, la priorita' e' incrementata (numeri alti corrispondono a bassa priorita') a seconda del valore o, se non specificato, viene assunto 10. Il nice di partenza e' di 20 per tutti. Utenti normali possono aumentare il nice (priorita' minore); solo l'operatore puo' diminuirlo per esempio con "nice -10".

Nohup esegue il comando immune da segnali di terminazione che arrivano dal terminale. Nohup dev'essere invocato dalla shell con "&".

ESEMPI

1) Si vuole dare un comando con un'esecuzione piuttosto lunga senza sovraccaricare il sistema e rallentare cosi' tutti gli utenti.

```
nice +15 <comando> &
```

In questo modo il sistema prosegue l'esecuzione del <comando> solo nei momenti in cui non e' gia' sovraccarico di lavoro, quando non ci sono molte altre richieste.

9.26. SHELL INTERATTIVA: VSH**NOME**

```
vsh - visual shell
```

SINTASSI

```
vsh
```

FUNZIONE

Vsh e' una shell altamente interattiva che facilita molte attivita' di programmazione. Per la sua stessa natura, e' opportuno che la si impari avendo il supporto pratico di un terminale. La maggior parte dei comandi constano di un solo carattere. Questi comandi hanno la funzione di visualizzare directory, richiamare gli editor, compilatori ecc...

Entrando in vsh viene visualizzata la directory corrente. Ogni file della directory e' marcato con una lettera. Per selezionarne uno basta dare la lettera corrispondente. Cosa succede dopo la selezione dipende dalla natura del file. Se il file e' una directory, vsh entra nella stessa. File di tipo testo vengono chiamati in editor.

Vsh visualizza fino a 20 files per volta. Directory piu' grosse vengono spezzate in pagine di 20 elementi. Vsh puo' contenere fino a 200 files per directory. Le pagine sono numerate da uno a dieci con 0 per selezionare la decima. Per selezionare una pagina basta digitare il numero corrispondente.

Una caratteristica di vsh e' la sua interfaccia con i compilatori e gli editor. Per iniziare una compilazione bisogna premere M. Vsh chiama il programma make. L'output di questa compilazione viene salvato per poter riesaminare gli errori di compilazione. Vsh numera ogni errore. E' possibile selezionare l'errore digitando il suo numero. Dopo questa selezione vsh chiama un editor sul file appropriato alla linea contenente l'errore. Si puo' alternarsi tra l'editor e vsh finche' si e' pronti per un'altra compilazione.

PARAMETRI

Vsh assume i seguenti parametri dai globali della shell con cui e' stato chiamato:

HOME La home directory. Questo parametro e' assegnato automaticamente dopo il collegamento al sistema.

SHELL

La shell di partenza. Quando e' necessario dare un comando shell, questo parametro seleziona il programma da usare.

TERM Il tipo del terminale. Il terminale deve avere l'indirizzamento del cursore.

SELEZIONE DI FILE

Un file si seleziona digitando la lettera corrispondente. A seconda del tipo del file, succedono cose diverse.

Tipo del file	Azione
Directory	Entra nella nuova directory
Archivio	Visualizza la directory di

	archivio
File oggetto	Visualizza la name list (simboli esterni)
Core	Chiama il debugger
Testo	Chiama l'editor
Altro	La selezione non avviene

OPERAZIONI SU DIRECTORY

Segue ora una lista di comandi ottenibili quando vsh mostra la directory.

<ctrl>D

Esce da vsh

<return>

Entra nella directory padre

a-t Seleziona il file corrispondente

1-9 Seleziona la pagina corrispondente

0 Seleziona la pagina 10.

+ Va alla pagina seguente

- Va alla pagina precedente

^ Va alla home directory

/ Va alla directory root

? Visualizza il file di aiuto per i comandi

! Esegue la shell, ritorna in vsh quando viene dato
<ctrl>D

% Come !

\$ Esegue la shell di Bourne (/bin/sh)

D Visualizza la data

I Visualizza un file. Vsh chiede il nome del file e lo visualizza considerandolo un testo ASCII. Cio' e' piu' veloce che entrare in editor sul file.

P Stato dei processi, chiama /bin/ps

W Esegue who

Y Esegue yank. Visualizza le porte libere

Questi sono comandi piu' complessi:

- F Seleziona il file per nome. Vsh chiede il nome del file e poi lo seleziona
- C Crea un nuovo file. Vsh chiede il nome ed il tipo del file. Si possono creare testi e directory, copiare e creare link tra files.
- L Elenca i files in formato esteso. Si ottengono le stesse informazioni di "ls -l"
- O Elenca le opzioni. Elenca il contenuto delle tavole dei comandi e dei parametri. Vsh chiede un comando che modifichi i parametri o che crei, modifichi o rimuova comandi.

RIMUOVERE FILES

Digitare R per entrare nella modalita' Remove. A questo punto, un file selezionato viene contrassegnato con "/" per essere rimosso. Il carattere * segna tutti i files della pagina. Nel caso che inavvertitamente abbiamo selezionato un file che non vogliamo cancellare, una seconda selezione dello stesso file fa in modo che non sia piu' segnato. Digitando R vengono rimossi i files marcati, <return> esce da Remove. Se vsh non puo' rimuovere un file (non e' di proprieta' nostra oppure e' una directory non vuota ecc...), visualizza il motivo ed aspetta un <return> per proseguire. La modalita' Remove accetta i seguenti comandi:

- <return>
Esce dalla modalita' Remove senza rimuovere nessun file
- <ctrl>D
Come <return>
- a-t Seleziona il file corrispondente
- * Seleziona tutti i files della pagina
- 0-9, +, -
Cambia pagina
- ? Visualizza il file di aiuto

MAKE, GREP E SHOWFILE

Vsh effettua collegamenti con make e grep, salva gli

output di questi processi e permette di esaminare i risultati nella modalita' Showfile. Sono disponibili i seguenti comandi:

- G Grep. Vsh chiede il pattern ed il nome del file su cui verra' cercato. Il risultato viene salvato nel file .grepout e visualizzato nella modalita' Showfile.
- S Visualizza il precedente grep.
- M Make. L'esecuzione di make e' controllata dal makefile della corrente directory. L'output viene visualizzato su video e contemporaneamente salvato nel file .makerror. Quando make termina, vsh visualizza .makerror nella modalita' Showfile.
- N Make viene eseguito indipendentemente. L'output e' salvato in .makerror ma non viene visualizzato. Quando make ha terminato, viene emessa 2 volte una segnalazione acustica. Il comando E ci permette di rivedere l'output.
- E Rivede gli errori dell'ultimo make nella modalita' Showfile

MODALITA' SHOWFILE

Showfile visualizza i risultati di make e grep. Viene chiamato dai comandi E ed S. Showfile e' chiamato automaticamente nel corso dei comandi M e G. Essenzialmente e' un editor speciale. Ha comandi che permettono di visualizzare un file. Si puo' inoltre esaminare una linea particolare del file. Showfile cerca questa linea ed estrae il nome del file e il numero di linea. Poi Showfile chiama l'editor cominciando sulla linea specificata.

Questi sono i comandi di Showfile. Non e' necessario trasmettere <return> alla fine del comando.

<numero>p

Stampa il file partendo dalla linea specificata.

<numero>e

Esamina la linea indicata cercando un file name e numero di linea. Se e' stato trovato, chiama l'editor su quel file.

<return>

Visualizza le prossime 12 linee

```
<numero><return>
    come <numero>e
```

```
<ctrl>D, q
    Esce da showfile
```

```
?    Visualizza il file di aiuto
```

Sono inoltre disponibili i comandi a livello della pagina della directory (a parte L ed R). Particolarmente utili sono i comandi M, G ed F per entrare in editor su un file che non e' stato trovato.

COMANDI DI CONTROLLO

Le operazioni di vsh sono controllate dalle proprie tavole di parametri e comandi. Il contenuto delle tavole puo' essere visualizzato con il comando O. Esse possono essere modificate attraverso comandi di controllo. Quando vsh e' chiamato, cerca nella home directory un file chiamato .vshrc. I comandi di controllo vengono letti da questo files, potendo cosi' personalizzare la vsh. Anche il comando O permette questa cosa.

I comandi di controllo devono presentarsi con la stessa grammatica lessicale dei comandi shell. Spazi e tabs separano le parti del comando. Le macro (come \$HOME) non sono disponibili. Il file /usr/ucb/lib/dflt.vshrc contiene i parametri di default. Il .vshrc personale e' appeso alla fine di dflt.vshrc.

Per modificare un parametro dev'essere usato il formato

```
nome-parametro    valore-del-parametro
```

Il nome del parametro puo' essere ognuno dei seguenti:

editor	Editor voluto. Per default e' /usr/ucb/bin/vi
nm	Programma di namelist. Per default e' /bin/nm
db	Programma di debug. Per default e' /bin/adb
make	Il programma di make. Per default e' /bin/make
grep	Il programma di grep. Per default e' /bin/grep

ar Il programma di archivio. Per default
 e' /bin/ar

Per esempio possiamo utilizzare l'editor ed anziche' vi
con

editor /usr/ucb/bin/ed

oppure eseguire grep con l'opzione "-v" con

grep '/usr/grep -v'

Gli apici sono necessari per comunicare che si tratta
dello stesso argomento (altrimenti il blank prima di
-v viene interpretato come separatore di argomenti).

Per definirsi un comando si usa il formato

<carattere> <parola-chiave> [parametro ...]

<carattere> e' il carattere con cui si chiama il nuovo
comando. Sono caratteri validi ! " # \$ % & ' () : * =
^ ~ [] { } < > , . / ? ed A-Z.

Queste sono parole-chiave valide:

date	Visualizza la data
showerror	Visualizza gli errori del precedente make
showgrep	Visualizza l'output del precedente grep
file	Seleziona un file. Se presente, viene selezio- nato il parametro, altri- menti vsh chiede il nome del file
home	Va alla home directory
grep	Esegue grep. Vsh chiede i parametri
wmake	Esegue make. Aspetta la fine
fmake	Esegue make senza aspettare
remove	Entra in modalita' remove
longlist	Stampa la lista in formato esteso dei file.

<code>display</code>	Visualizza il contenuto di un file. Se non sono presenti parametri vsh chiede il nome del file
<code>options</code>	Visualizza le tavole dei parametri e dei comandi.
<code>exec</code>	Esegue un comando direttamente tramite la chiamata di sistema <code>exec</code> . Il primo parametro deve contenere il nome del programma. E' necessario dare il path name completo di un comando. Ogni successivo parametro diventa un parametro del comando. Ridirezioni e macro sono disponibili.
<code>create</code>	Vsh chiede un nuovo file name. Il file viene creato.
<code>null</code>	Cancella i comandi.

Per esempio, e' possibile cambiare il comando della `longlist` da `L` a `<ctrl>L` cosi' procedendo: si digita "0" e

```
<ctrl>L longlist
L null
```

quando ci compare la richiesta `Parm:`. Viene assegnato cioe' a `<ctrl>L` il comando di `longlist` e viene cancellato il comando `L`. Se invece si vuole avere l'opportunita' di eseguire il comando `du` all'interno di `vsh`, richiamandolo con `T`, bisogna dare

```
T exec /usr/ucb/bin/du
```

ESEMPI

Questo e' un esempio di file `.vshrc`

```
, exec /bin/who
= display
grep /usr/ucb/bin/grep
! exec /usr/ucb/bin/csh -f
T exec /usr/ucb/bin/du .
```

3.27. ALTRI COMANDI

I comandi descritti fino ad adesso sono solo una parte dei comandi UNIX. Nel sistema con cui lavoriamo sono presenti circa 400 comandi. E' chiaramente impossibile trattarli tutti, non e' neanche necessario saperli usare tutti. Per molte di essi, l'importante e' sapere che esistono, cosa all'incirca fanno e dove trovare documentazione al momento in cui servono. Si rimanda per questo ai manuali reperibili in linea con il comando man ed ai manuali cartacei di UNIX. La comprensione di tutto cio' descritto finora permette di fare un salto qualitativo nella conoscenza di UNIX.

Questa e' una sintesi di altri comandi UNIX:

adb Debugger interattivo.

apropos

Chiarisce velocemente la funzione di un comando

ar Mantiene archivi e librerie

banner

Stampa una scritta in caratteri grossi.

bas Basic

bc Linguaggio aritmetico ad infinita precisione. Utile anche per calcoli tipo calcolatore tascabile.

cal Stampa un calendario

chown, chgrp

Cambia proprieta' e gruppo di un file

chsh Cambia la shell con cui ci colleghiamo

col Filtro per i reverse line feed. Viene usato in collegamento attraverso una pipe con nroff per adattare il testo a stampanti con scrittura bidirezionale.

crypt

Crittografa files. Usato per protezione dei files.

f77 Compilatore Fortran 77

ld Loader. Permette di riunire piu' files in formato rilocabile in uno di tipo assoluto.

leave Ricorda il momento in cui ci si deve scollegare

ln Crea links tra files

od Dump ottale di un file

prep Prepara un testo per attivita' statistiche. Ogni riga contiene una parola dell'input.

rev Inverte i caratteri di ogni linea di un file.

strip Rimuove simboli e parti rilocabili da un file binario.

tbl Formatta tabelle per nroff o troff

time Calcola il tempo di un comando. Tempo di CPU, di sistema e tempo reale.

touch Aggiorna la data di ultima modifica di un file

umask Determina le protezioni nella creazione di un file

CAPITOLO 4

Shell e cshell

4.1. LA SHELL

Sinora abbiamo visto che la shell e' un interprete di comandi. Tuttavia le possibilita' offerte da tale interprete sono tali da permetterne l'uso come linguaggio di programmazione. E' cosi' possibile creare un proprio comando mettendo una serie di comandi elementari in un file. Cambiando poi il flag di eseguibilita' (con `chmod +x <nomefile>`) si puo' richiamare la sequenza come un qualsiasi altro comando di UNIX.

4.1.1. Parametri posizionali

La forma generale di un comando UNIX e'

```
<comando> [<arg1> <arg2> . . . <argn> ]
```

Se vogliamo rendere omogenea l'interfaccia tra l'utente e il programma shell dobbiamo poter accedere in qualche modo ai parametri con cui il comando e' stato richiamato. Per accedere ai parametri posizionali la shell mette a disposizione delle variabili, chiamate "parametri

posizionali" che vengono assegnate alla chiamata della procedura. Queste variabili sono : \$1 per il primo parametro, \$2 per il secondo, \$3 per il terzo, e così via fino ad esaurire il numero dei parametri. Es: Supponiamo di voler scrivere un comando chiamato "chissasece" il cui scopo è quello di dirci se un utente è collegato o no. Scriveremo in un file:

```
who | grep $1
```

Lo rendiamo eseguibile con

```
chmod +x chissasece
```

e quando daremo il comando

```
chissasece gianni
```

è come se avessimo scritto `who | grep gianni` che darà come risultato:

```
gianni      tty3      May 5 12:00
```

se gianni è collegato oppure niente se gianni non è collegato. Supponiamo adesso di voler scrivere un comando che compili un programma in pascal che si trova nel file specificato nel primo argomento e produca un eseguibile il cui nome sia il secondo parametro. Creiamo il comando "PI" con il seguente contenuto:

```
pi $1
mv obj $2
```

lo rendiamo eseguibile con

```
chmod +x PI
```

e, se prova.p è un nostro programma in pascal, proviamo

```
PI prova.p prova
```

se tutto è andato bene dovremmo avere in "prova" il risultato della nostra compilazione. Possiamo anche "personalizzare" i nostri comandi aggiungendo dei messaggi, nell'esempio precedente potremmo aggiungere una terza linea.

```
echo $1 compilato. L'eseguibile si chiama $2
```

Riprovando il nostro comando

```
PI prova.p prova
```

al termine della compilazione verra' visualizzato il messaggio:

prova.p compilato. L'eseguibile si chiama prova.

4.1.2. Strutture di controllo - for -

In alcuni casi il numero dei parametri specificati non e' noto a priori; puo' essere noioso riscrivere la stessa sequenza di comandi per ciascun parametro posizionale. Vogliamo ad esempio costruire un comando che ci permetta di reperire il numero telefonico di una serie di nominativi dati come parametri; la procedura risultante sara' simile a questa:

```
for nome
do grep $nome tele-elenco
done
```

for...done e' una struttura che permette di eseguire tante volte quanti sono i parametri le istruzioni contenute tra for e done sostituendo ad ogni iterazione il valore del prossimo parametro.

Scriviamo l'esempio appena fatto in un file "chenum" e rendiamolo eseguibile. creiamo il file "tele-elenco" in questa forma:

```
Gianni Mori 4481907
Filippo Bianchi 4144263
Ivano Fossati 8159354
Marina Rossi 4950266
Ileana Rossi 525087
```

proviamo ora questo comando:

```
chenum Rossi Bianchi
```

dopo qualche istante avremo:

```
Marina Rossi 4950266
Ileana Rossi 525087
Filippo Bianchi 4144263
```

A questo punto possiamo creare il comando "aggiungi", che aggiunga un numero telefonico alla nostra rubrica cosi' strutturato:

```
echo $* >>tele-elenco
```

La variabile "\$*" contiene tutti i parametri (escluso il nome del comando stesso) forniti ad una procedura in shell. Se diamo:

```
aggiungi Alberto Abba 923540
```

aggiungeremo un nominativo al nostro elenco telefonico senza ricorrere ad un editor. La forma piu' generale del comando "for" e' questa:

```
for <nome> [in <parola1 parola2 . . . >]
do <elenco di comandi >
done
```

dove

<nome> e' una qualsiasi sequenza di lettere cifre e "_" (underscore) comincianti con una lettera, e

<elenco di comandi> e' una qualsiasi sequenza di comandi separati da un a capo o da ";".

Da notare che "do" e "done", se e' presente la parte "in <parola1 ...>", per essere riconosciuti devono comparire come primi caratteri non bianchi della linea o essere preceduti da ";" altrimenti "do" non deve essere preceduto da ";". In parole povere se in chenum avessimo scritto :

```
for i ; do grep $i tele-elenco ; done
```

non avrebbe funzionato perche' l'elenco di parole e' assente mentre c'e il ";" per funzionare dovrebbe essere fatto in questo modo:

```
for i do grep $i tele-elenco ; done
```

oppure cosi'

```
for i
do
grep $i tele-elenco
done
```

che sono entrambe forme corrette.

Adesso vogliamo scrivere un comando che visualizzi i nomi di tutte le procedure e funzioni dei programmi in pascal che sono nella directory attuale, preceduti dal numero di linea a cui si trovano. E' necessario usare un "for" per scandire tutti i parametri in pascal (che sappiamo devono avere il suffisso ".p") e per ognuno scriviamo il nome, l'elenco

delle procedure e delle funzioni. Il risultato e' il comando seguente:

```
for prog in *.p
do
    echo +++++ $prog +++++
    echo
    echo - procedure -
    grep -n procedure $prog
    echo
    echo - funzioni -
    grep -n function $prog
    echo
done
```

l'opzione -n di grep fa precedere ogni linea visualizzata dal numero di linea.

4.1.3. Strutture di controllo - Case -

Se vogliamo applicare una serie di comandi a ciascuno dei parametri posizionali, la struttura "for" ci viene in aiuto risolvendo brillantemente il problema. Se vogliamo far dipendere il comportamento del comando dal numero dei parametri o dal tipo dei parametri trasmessi (ad esempio per poter dare delle opzioni) dobbiamo avvalerci di un'altra struttura che la shell ci mette a disposizione: il "case". Ecco il comando append:

```
case $# in
    1) cat >> $1;;
    2) cat $1 >> $2;;
    *) echo 'uso: append [ da ] a' ;;
esac
```

Spiegazione:

La variabile \$# contiene il numero di parametri forniti al comando. Se il numero di argomenti e' uguale a 1 allora appende lo standard input all'unico argomento, se \$# e' uguale a 2 allora appende il contenuto del file specificato come primo argomento al secondo, altrimenti (* ha il suo solito significato : qualsiasi sequenza di caratteri) viene emesso un messaggio di delucidazione circa l'uso del comando. Generalmente si usa un asterisco per poter intercettare l'esecuzione e mantenerne il controllo in caso di argomenti illegali o

imprevisti.

La struttura case puo' essere usata per il trattamento delle opzioni in un comando. Ad esempio per costruire un comando che trasferisca il contenuto dei suoi argomenti su terminale o su stampante a seconda che sia presente o meno l'opzione "-p"; se il trasferimento e' diretto alla stampante ogni file sara' preceduto dal suo nome, altrimenti viene eseguito il comando more:

```
case $1 in
-p) shift
    for i
    do
        echo '===== '>tmp.$$
        echo $i >>tmp.$$
        echo '===== '>>tmp.$$
        echo >> tmp.$$
        cat $i >> tmp.$$
    done
    cat tmp.$$ | lpr
    rm -f tmp.$$
*) more $$
esac
```

Spiegazione:

Se il primo parametro e' uguale a "-p" allora vengono riassegnati tutti i parametri posizionali (shift assegna \$2 a \$1, \$3 a \$2, e cosi' via eliminando \$1), viene scritto su un file temporaneo il nome del file incorniciato, poi "cat \$1 >> tmp.\$\$" accoda al file temporaneo il file attuale, per ogni argomento del comando; quando l'elenco dei files da trasferire e' esaurita il file temporaneo viene mandato in stampa e rimosso. La variabile "\$\$" e' una variabile di shell predefinita che contiene il numero di processo della shell attuale (quella che sta eseguendo questo comando). L'opzione -f del comando rm fa si che non vengano emessi messaggi di errore (ad esempio perche' il file non esiste).

Vogliamo provare adesso a costruirci un comando che compila i suoi argomenti (anche scritti in linguaggi diversi) usando il compilatore adatto. La scelta del compilatore da usare e' fatta in base al suffisso del file:

```
.p      pascal
.s      assembler
.c      C
.b      basic
```

vediamo il comando "compila"

```
for file
do
    case $file in
        *.p) pi $file;;
        *.c) cc $file;;
        *.s) as $file;;
        *.b) bas $file;;
        *) echo $file : suffisso sconosciuto;;
    esac
done
```

Sarebbe utile poter cambiare automaticamente il nome del nostro programma compilato in quello del programma sorgente privato del suffisso, ad esempio avere l'assoluto di prova.c in prova, di fattoriale.p in fattoriale. Dovremmo cioè poter estrarre dal nome di un file una certa parte terminale. Naturalmente ciò è possibile ed il comando che assolve a questo compito si chiama "basename". Vediamo subito qualche esempio :

```
basename pippo.p .p
```

produce:

```
pippo
```

mentre

```
basename mio.prog .prog
```

produce:

```
mio
```

Rimane il problema di poter assegnare il risultato di basename a una qualche variabile di shell. Ciò è possibile usando la struttura

```
nomefile=`basename fattoriale.p .p`
```

la shell esegue l'elenco di comandi che si trova tra i due ``" quindi sostituisce i comandi con il loro output. Quindi con la struttura precedente avremo "fattoriale" nella variabile nomefile.

Riscriviamo ora il comando "compila" usando quello che abbiamo appena imparato :

```
for file
do
    case $file in
        *.p|*.c|*.s|*.b) case $file in
```

```

*.p) obj=`basename $file .p`
    pi $file
    mv obj $obj;;
*.c) obj=`basename $file .c`
    cc $file
    mv a.out $obj;;
*.s) obj=`basename $file .s`
    as $file
    mv a.out $obj;;
*.b) obj=`basename $file .b`
    bas $file
    mv a.out $obj;;
esac;;
*) echo $file: suffisso sconosciuto;;
esac
done

```

Spiegazione:

Per ogni argomento del comando (for file), se il nome del file termina in ".p", ".c", ".s" o in ".b" (il carattere "i" e' l'or). allora si prosegue nel determinare qual'e' il suffisso di nomefile che stiamo trattando per mezzo di "basename" e lo si mette nella variabile "obj". Poi si richiama compilatore adatto e si cambia il nome di default dell'eseguibile (obj per il compilatore pascal pi ed a.out per gli altri linguaggi) prodotto dal compilatore in quello del file sorgente privato del suffisso contenuto nella variabile "obj", altrimenti viene emesso un messaggio di errore.

Oltre a "i" per separare alternative che rientrano nello stesso ramo di un case esiste la forma "[<caratteri>]" per indicare uno qualsiasi dei caratteri racchiusi tra parentesi quadre([abd] sta per a oppure b oppure d), la forma [<carattere>-<carattere>] indica un qualsiasi carattere nell'intervallo ([a-d] significa a oppure b oppure c oppure d), e "?" per indicare un qualsiasi carattere. Si puo' evitare il significato particolare di "!", "?", "[", "]", "*", facendoli precedere dal carattere \. La forma generale di case e' la seguente:

```

case <parola> in
    <pattern>) <elenco di comandi>;;
    . . .
esac

```

dove

<parola> e' una qualsiasi sequenza di caratteri. Parola e' soggetta a sostituzione di variabili. <pattern> e' una sequenza di caratteri composta come abbiamo visto

da eventuali metacaratteri

<elenco di comandi> e' una serie di comandi separati da ";" o da una nuova linea.

4.1.4. Le variabili di shell

Sinora abbiamo usato le variabili di shell senza approfondirne molto il l'uso e il significato.

Le variabili di shell sono sempre considerate delle stringhe; il nome di una variabile comincia con una lettera e consiste di lettere cifre e di underscore. E' possibile assegnare un valore a delle variabili scrivendo ad esempio

```
nome=fabio citta=milano cap=20100
```

In questo caso assegnamo i valori "fabio", "milano", "20100" rispettivamente a "nome", "citta", "cap"; una variabile puo' essere utilizzata antecedendo il suo nome con il carattere "\$". Possiamo ottenere il valore in ogni momento scrivendo ad esempio:

```
echo $nome $citta $cap
```

Si puo' assegnare la stringa nulla scrivendo ad esempio,

```
niente=
```

Una forma piu' generale per indicare il valore di una variabile e' \${nome} che e' equivalente a \$nome e che e' usata quando il nome della variabile e' seguito da lettere o cifre. Per esempio, se la variabile "nome" contiene "fabio", il comando

```
echo ${nome}a
```

da come risultato

```
fabioa
```

mentre se avessimo scritto

```
echo $nomea
```

il risultato sarebbe stato il valore eventuale della variabile "nomea" che per default viene assunto essere la stringa nulla. D'ora in poi ci riferiremo al processo di

sostituzione delle variabili con il termine "sostituzione di variabili". Infatti la shell opera come un macro espansore, cioè sostituisce ad una stringa di caratteri preceduta da "\$" l'eventuale valore della variabile in questione.

Parleremo in seguito dell'ordine in cui le varie sostituzioni (sostituzione di variabili, sostituzione di comandi, generazione dei nomi dei files) sono attuate dalla shell.

4.1.5. Stato di uscita di un comando - test

Quando un comando è terminato la shell pone in \$? il codice di terminazione; tale codice è un numero interno che indica come è terminato il comando. È possibile così sapere ad esempio se c'è stato qualche errore di esecuzione in un programma o comunque se è terminato normalmente. Se un programma ritorna 0 allora è terminato normalmente, altrimenti (in genere si usa -1 o 1) è avvenuto un errore di qualche tipo. Ad esempio scrivendo

```
cat
<delete>
echo $?
```

avremo 1 perché il programma cat è terminato in modo anormale. Otteniamo lo stesso risultato dando

```
rm cap.3
```

se non esiste il file "cap.3".

Il comando "test" esegue una serie di controlli sui suoi argomenti ritornando uno stato di uscita uguale a zero se la condizione è verificata. Qui di seguito sono elencate le opzioni più usate ed il loro significato.

```
-r file   :vero se il file esiste ed è leggibile.
-w file   :vero se il file esiste ed è scrivibile.
-f file   :vero se il file esiste e non è una directory.
-d file   :vero se il file esiste ed è una directory.
-s file   :vero se il file esiste e non è vuoto.
s1 = s2    :vero se le stringhe s1 e s2 sono uguali.
s1 != s2   :vero se le stringhe s1 e s2 sono diverse.
s1         :vero se s1 non è la stringa nulla.
```

Per un corretto funzionamento è necessario separare gli argomenti da almeno uno spazio. Ad esempio se dobbiamo confrontare una variabile con una stringa, una scrittura del

tipo

```
test $nome="pippo"
```

dara' sicuramente luogo ad un errore o comunque ad un risultato che non e' quello del confronto della variabile "nome" con la stringa "pippo".

4.1.6. Strutture di controllo - if

La struttura "if" permette di subordinare l'esecuzione di un'elenco di comandi al buon esito di altri comandi. La forma generale e':

```
if <elenco di comandi>
then <elenco di comandi>
[<parte else>]
fi
```

dove <parte else> e':

```
else <elenco di comandi>
```

oppure

```
elif <elenco di comandi>
then <elenco di comandi>
```

Ma vediamo subito un esempio applicato ad una procedura di shell.

```
if test $# = 2
then echo "2 argomenti"
elif test $# -gt 2
then echo "piu' di 2 argomenti"
fi
```

Spiegazione:

Se il numero di argomenti ricevuti e' uguale a 2, allora scrive "2 argomenti" altrimenti se (elif) il numero di argomenti e' maggiore di 2 (l'opzione -gt di test significa "maggiore di"), allora scrive "piu' di 2 argomenti".

In unione al comando test la struttura "if" diventa uno strumento molto potente per l'esecuzione controllata di

comandi. Proviamo a rivedere qualcuno dei comandi che abbiamo già scritto in modo da migliorarne le funzionalità:

```
for file
do
    if test -f $file
    then
        case $file in
            *.p|*.c|*.s|*.b) case $file in
                *.p) obj='basename $file.p'
                    pi $file
                    mv obj $obj;;
                *.c) obj='basename $file.c'
                    cc $file
                    mv a.out $obj;;
                *.s) obj='basename $file.s'
                    as $file
                    mv a.out $obj;;
                *.b) obj='basename $file.b'
                    bas $file
                    mv a.out $obj;;
            esac;;
            *) echo $file: suffisso sconosciuto;;
        esac
    else echo $file non esiste.
    fi
done
```

Spiegazione:

Il controllo effettuato con "if test -f \$file" ci permette di stabilire se il parametro che stiamo esaminando in questo momento esiste o no; in caso affermativo si procede regolarmente, altrimenti viene emesso un messaggio di errore che indica che non esiste il file specificato.

4.1.7. Strutture di controllo - until e while -

La shell mette a disposizione anche due strutture per il controllo delle iterazioni: until e while; il primo esegue una serie di comandi almeno una volta ed esce per condizione vera; il secondo esegue il controllo prima di iniziare la sequenza di comandi ed esce per condizione falsa. La sintassi esatta di queste strutture è questa:

until:

```
until <elenco di comandi>
do
<elenco di comandi>
done
```

Spiegazione:

Vengono eseguiti i comandi compresi tra "do" e "done" dopo di che vengono eseguiti i comandi tra "until" e "do" e controllato lo stato di uscita dell'ultimo comando eseguito, se diverso da zero l'esecuzione continua con gli eventuali comandi che seguono "done" altrimenti ritorna al punto di partenza.

while:

```
while <elenco di comandi>
do
<elenco di comandi>
done
```

Spiegazione:

L'elenco di comandi tra "while" e "do" e' eseguita; se lo stato di uscita dell'ultimo comando e' zero allora viene eseguita l'elenco di comandi tra "do" e "done" e si ricomincia da capo; altrimenti si prosegue con gli eventuali comandi che seguono "done"

Queste strutture sono usate soprattutto in procedure interattive, nelle quali cioe' il numero di iterazioni non e' conosciuto o comunque non dipende dal numero di argomenti del comando. L'uso interattivo della shell e' possibile grazie al comando "read" che permette di leggere dalla tastiera il valore da assegnare ad una o piu' variabili. La sintassi di questo comando "built-in" (vuol dire che questo comando e' interpretato ed eseguito direttamente dalla shell) e' questa:

```
read parola <parola>
```

In pratica, se noi scriviamo

```
read nome
```

la shell attendera' che noi scriviamo qualcosa terminato da un <return> dopo di che ci ripresentera' il solito segnale di pronto (prompt). Se adesso diamo:

```
echo $nome
```

otterremo cio' che abbiamo scritto. Se gli argomenti di "read" sono piu' di uno allora la riga letta verra' divisa in parole dopo di che la prima parola verra' assegnata alla prima variabile, la seconda alla seconda e cosi' via, se il numero di parametri fornito alla read e' maggiore delle parole date in input allora alle rimanenti viene assegnata la stringa nulla. Se il numero di parole dato in input e' superiore a quello dei parametri allora all'ultimo parametro viene assegnato la parte rimanente della riga di input.

Vediamo qualche comando di utilita' generale che sfrutti le nuove strutture che abbiamo appena visto. Proviamo a scrivere un comando di rimozione interattiva dei files:

```
for file
do
    echo -n $file '? '
    read risposta
    case $risposta in
        si) rm -f $file
            echo $file rimosso;;
        *) echo $file non rimosso;;
    esac
done
```

Spiegazione:

Le uniche cose che sinora non abbiamo ancora visto sono l'opzione "-n" di echo e l'opzione "-f" di rm. Normalmente echo va a capo dopo aver emesso l'ultimo argomento, l'opzione "-n" fa si che successive echo vadano a scrivere sulla stessa linea. Nel nostro caso il cursore si trovera' immediatamente dopo il punto di domanda in attesa di una risposta. L'opzione "-f" di rm fa si che il comando rm non dia in nessun caso un messaggio di errore, anche se il file che si tenta di rimuovere non esiste. Il punto di domanda viene messo tra apici per fargli perdere il significato speciale che ha per la shell. Altrimenti "?" verrebbe espanso con i nomi dei files (della directory corrente) lunghi un solo carattere. Il file viene cosi' effettivamente rimosso solo se la risposta e' "s" o "si".

4.1.8. Qualcosa di piu'

Ora che abbiamo visto le principali strutture di controllo possiamo approfondire ulteriormente alcune delle caratteristiche della programmazione della shell tra cui le strutture dati, le ridirezioni, la gestione degli errori e le variabili.

4.1.9. Le redirezioni

La shell, come abbiamo gia' visto, permette di ridefinire lo standard input e lo standard output di un comando, per mezzo dei simboli ">", "<", ">>". Oltre a queste sono disponibili altre forme di ridirezione, di cui una molto importante:

<<parola

Lo standard input e' preso dalle linee che seguono fino ad una linea che consiste solo di "parola". Se "parola" e' racchiusa tra apici o doppi apici allora le linee che seguono sono passate letteralmente al comando senza nessun tipo di sostituzione, in caso contrario queste linee sono soggette alla sostituzione di variabili e di comandi. In questo caso se si vuole introdurre uno dei caratteri "\$", "\", "\" bisogna farli precedere da "\". Questo modo di fornire l'input di un comando e' utile soprattutto nell'uso di "ed" all'interno di una procedura, permettendo di modificare il contenuto di un file in un modo dipendente dal valore assunto da certe variabili all'interno di una procedura. Sfruttando questa caratteristica proviamo a scrivere una procedura che completi "chenum" e "aggiungi" permettendo di rimuovere (se viene data l'opzione "-d") o di modificare un dato all'interno di tele-elenco:

```
case $# in
  [01]) echo 'uso: cambia [-d dato] [dato nuovodato]';;
  2) case $1 in
      -d)ed - tele-elenco<<?ine
        /$1/d
```

```

fine
    ;;
    *) ed - tele-elenco<<fine
    /$1/s//$2/
fine
    ;;
    esac;;
esac;;

```

2>file

Lo standard error viene ridiretto su "file".

>&- Lo standard output viene chiuso. Serve quando si esegue un comando non per cio' che produce su output ma per i suoi effetti collaterali. Ad esempio per eliminare messaggi non desiderati come il "?" presentato da "ed" quando una ricerca fallisce.

2>&-

Lo standard error viene chiuso. Si eliminano cosi' tutti i messaggi di errore di un comando.

4.1.10. Parametri posizionali

E' possibile assegnare dei valori ai parametri posizionali all'interno di una procedura con l'istruzione:

```
set <elenco di valori>
```

Se noi scriviamo ad esempio:

```
set *
```

assegneremo ai parametri posizionali i nomi di tutti i files nella directory corrente (tranne quelli che cominciano per punto); con l'istruzione

```
set `date`
```

i parametri posizionali conterranno i vari campi del comando "date" e sara' cosi' possibile personalizzare il comando cambiando l'ordine di presentazione dei vari campi; un successivo:

```
echo $3 $2 $6
```


genera

4 Mar 1983

4.1.11. Variabili

Oltre a quelle che abbiamo già visto, esistono altre variabili che vengono inizializzate dalla shell:

\$- Contiene le opzioni della shell correntemente definite (come `-v` e `-x`).

\$MAIL

Quando è usata interattivamente, la shell prima di presentare il segnale di pronto controlla se il file specificato da questa variabile è stato modificato e in caso affermativo viene presentato il messaggio "You have mail.". Tipicamente questa variabile è inizializzata nel file ".profile" situato nella home directory. Può essere assegnata con

```
MAIL=/usr/spool/mail/paolo
```

\$HOME

È il path name della home directory. È l'argomento sottinteso del comando "cd". Scrivere `cd` equivale a scrivere `cd $HOME`. Il nome della directory corrente è usato come prefisso per risolvere riferimenti a file il cui nome non cominci per "/", la directory corrente può essere cambiata per mezzo del comando "cd".

\$PATH

È una lista di directory (cammino di ricerca) che contengono comandi. Ogni volta che un comando viene richiamato, le directory presenti nell'elenco vengono scandite alla ricerca di un file eseguibile che abbia il nome cercato. Se la variabile `$PATH` non esiste allora i comandi vengono ricercati nella directory corrente, in `/bin` e in `/usr/bin`, altrimenti `$PATH` consiste di nomi di directory separati da ":". Per esempio:

```
PATH=:/usr/paolo/bin/:/bin:/usr/bin
```

indica che i comandi vanno cercati nella directory corrente (indicato dal ":" iniziale), in /usr/paolo/bin, in /bin e in /usr/bin. L'utente puo' modificare il contenuto di questa variabile per far si' che i comandi vengano cercati anche in una propria directory indipendentemente dalla directory corrente. Se il nome del comando contiene un "/" allora il "cammino di ricerca" non viene usato e viene semplicemente effettuato un tentativo di eseguire il comando. Normalmente questa variabile viene inizializzata in ".profile".

\$PS1 La stringa di caratteri che la shell presenta quando e' pronta a ricevere un comando. Per default e' "\$ ".

\$PS2 La stringa di caratteri che la shell presenta quando e' in attesa di ulteriore input (es: apici non chiusi). Per default e' "> ".

\$IFS L'insieme di caratteri che vengono assunti come separatori di parole. Non e' consigliabile modificare il valore di questa variabile.

4.1.12. Ordine di valutazione di un comando

La shell esegue come abbiamo visto una serie di sostituzioni sugli argomenti dei comandi, vedremo ora l'ordine in cui queste sostituzioni vengono effettuate e il modo per inibirle. Prima che un comando venga eseguito vengono effettuate le seguenti sostituzioni in questo ordine:

Sostituzione di variabili (ad esempio \$nome viene sostituita con il suo valore).

Sostituzione di comandi (ad esempio `pwd` viene sostituito con il path name della directory corrente).

Suddivisione in parole. Parole vengono assunte essere sequenze di caratteri separati da almeno uno dei caratteri presenti nella variabile predefinita \$IFS, che per default contiene uno spazio, un <tab> e il carattere di nuova linea. La stringa nulla non viene considerata una parola se non e' racchiusa tra apici o doppi apici. Quindi se diamo:

```
echo $aa
```

se il valore della variabile "aa" e' la stringa nulla, avra' l'effetto di chiamare echo senza parametri; mentre

```
echo ''
```

chiamera' echo con la stringa nulla come paramero.

Generazione di nomi di file (viene generata una lista ordinata alfabeticamente dei nomi di files che corrispondono al modello indicato). Ciascun nome costituisce un argomento separato. Es: se i files nella direttrice corrente sono:

```
pippo    prova.p    lettere    mbox
```

allora un "echo p*" dara' come risultato

```
pippo prova.p
```

cioe' tutti i nomi di files che cominciano con "p".

Queste valutazioni vengono effettuate anche per le parole associate a "for", mentre per "case" viene effettuata solo la sostituzione di variabili. Ad esempio si possono eseguire delle azioni sui files presenti nella directory corrente con:

```
for i in `ls`
do
  <elenco di comandi>
done
```

oppure con

```
for i in *
do
  <elenco di comandi>
done
```

Invece una istruzione del tipo

```
case `ls` in
  <elenco di comandi>
esac
```

non e' corretta o, per lo meno, non ottiene l'effetto sperato perche', come abbiamo detto, la parola che determina la selezione e' soggetta solo alla sostituzione di variabili e non a quella di comandi.

4.1.13. Prevenzione dell'espansione

Quando si impartisce un comando la shell per prima cosa interpreta i caratteri speciali come il simbolo di "pipe" ("|") e quelli di ridirezione creando se necessario i files indicati, dopo di che esegue le sostituzioni che abbiamo visto. E' importante notare che le parole che indicano il nome del file nelle redirezioni sono soggette solo alla sostituzione di variabili e di comandi, non viene quindi effettuata la suddivisione in parole e, ovviamente, la generazione di nomi di files. Si puo' inibire il significato particolare di un qualsiasi singolo carattere facendolo precedere da "\", ad esempio

```
echo
```

dara' come risultato semplicemente un asterisco.

Se una qualsiasi sequenza di caratteri e' racchiusa tra apici allora questa costituisce una singola parola, ed inoltre viene inibita qualsiasi sostituzione.

Se una qualsiasi sequenza di caratteri e' racchiusa tra doppi apici allora su questa stringa viene inibita la suddivisione in parole separate e la generazione di nomi di files ma non la sostituzione di comandi e quella di variabili; ad esempio

```
echo "\ls"
```

passera' ad "echo" un singolo parametro costituito dal risultato del comando "ls";

```
echo "$@"
```

passera' come unico argomento ad "echo" i parametri posizionali ed e' equivalente a

```
echo "$1 $2 ..."
```

A questo proposito la notazione "\$@" e' equivalente a "\$*" tranne quando e' racchiusa tra doppi apici, nel qual caso e' equivalente a

```
echo "$1" "$2" ...
```

Nel caso sia desiderata piu' di una valutazione si deve ricorrere al comando "eval" che valuta i suoi argomenti e li passa a sua volta da valutare alla shell. Per esempio se il

valore della variabile "a" e' "\$b" e il valore della variabile "b" e' "ciao", allora

```
echo $a
```

dara' semplicemente "\$b", mentre

```
eval echo $a
```

dopo la valutazione effettuata da "eval" diventera'

```
echo $b
```

che eseguito dara'

```
ciao
```

4.1.14. Gestione degli errori

Gli errori che possono incorrere durante l'esecuzione di una procedura di shell sono di vario tipo. Alcuni di questi provocano l'interruzione dell'esecuzione della procedura; mentre altri fanno si' che la shell passi ad eseguire il comando successivo. Le azioni intraprese dalla shell quando viene identificato un errore dipendono oltre che dal tipo di errore anche dal fatto che la shell sia interattiva o meno (una shell e' interattiva se il suo input e il suo output sono connessi ad un terminale, oppure se e' stata chiamata con il flag "-i"). Se una shell e' chiamata con il flag "-e" allora qualsiasi errore causa la sua terminazione (anche se interattiva). I seguenti errori causano la terminazione di una procedura di shell (se interattiva la shell passera' alla lettura del prossimo comando):

- (1) Errori di sintassi.
- (2) Una interruzione dal terminale.
- (3) Fallimento di uno dei comandi residenti della shell come "cd".

Altri errori faranno si che la shell passi all'esecuzione del comando successivo. A volte non e' desiderabile che una procedura di shell sia interrompibile (ad esempio per mezzo del tasto <delete>), specialmente se vengono eseguite operazioni delicate come, ad esempio, la copia di un file, oppure si vuole poter mantenere il controllo dell'esecuzione (ad esempio per rimuovere files temporanei). Il comando

"trap" permette di specificare l'azione da intraprendere al ricevimento di un particolare segnale. La shell riceve un segnale di interruzione. La sua sintassi e' questa:

```
trap '[azioni]' <lista di segnali>
```

<azioni> puo' essere o la stringa nulla, nel qual caso il segnale e' ignorato, oppure una lista di comandi, che verranno eseguiti al ricevimento dei segnali specificati.

<lista di segnali> e' costituita da uno o piu' numeri compresi tra uno e quindici.

Gli unici segnali che ci interessano sono 2 ("interrupt") e 3 ("quit") e sono gli unici che possono essere inviati da terminale. Il primo viene inviato a tutti i processi attivati dal terminale da cui e' partito il segnale quando viene premuto il tasto "delete" o "del" e provoca la terminazione di quei processi che non hanno specificato una particolare azione, il secondo (trasmesso con <ctrl>\) provoca anche (sempre se non e' stata specificata nessuna azione) la creazione di un file "core" che contiene l'immagine della memoria al momento dell'interruzione. E' possibile analizzare il contenuto di questo file per mezzo del comando "adb" (non e' comunque una operazione alla portata di tutti, ed e' consigliabile rimuovere sempre questo file dato che solitamente e' di dimensioni piuttosto consistenti). Se in una procedura mettiamo la linea:

```
trap 'echo programma interrotto; exit' 2
```

quando tentiamo di interrompere la procedura in corso verra' emesso il messaggio

```
programma interrotto
```

4.2. LA CSHELL

La cshell deve il suo nome al fatto che la sintassi di alcuni suoi comandi e' simile a quella di corrispondenti istruzioni del linguaggio di programmazione "C". La caratteristica principale della cshell e' quella di facilitare al massimo il suo uso da terminale. In questo capitolo ci limiteremo ad illustrare appunto quelle caratteristiche che rendono agevole l'uso della cshell da terminale e tratteremo brevemente la sua programmazione.

4.2.1. Parole

La cshell divide le linee di ingresso in parole considerando come separatori lo spazio e il carattere di tabulazione con le seguenti eccezioni. I caratteri "&" "|" ";" ">" "<" "(" ")" formano parole separate. Le coppie di caratteri "||", "<<", ">>", formano singole parole. Queste parole particolari possono entrare a far parte di altre parole o perdere il loro significato particolare se sono precedute da "\". Un <return> preceduto da "\" e' equivalente ad uno spazio. Una sequenza di caratteri compresa tra doppi apici o tra "'" o tra "\"" formano parti di parola (a" c e ff" qq e' una sola parola).

4.2.2. Variabili

La cshell mantiene un insieme di variabili ciascuna delle quali ha come valore una lista di zero o piu' parole. Alcune di queste variabili sono "settate" o utilizzate dalla shell. Per esempio la variabile "argv" e' un'immagine della lista di argomenti della shell. La variabile "prompt" contiene la stringa che la shell emette quando e' in attesa di un comando, per difetto vale "% " per un utente normale e "# " per l'operatore. I valori delle variabili possono essere visualizzati o cambiati per mezzo dei comandi "set" e "unset". Si puo' assegnare un valore ad una variabile scrivendo ad esempio:

```
set nome=Filippo
```

oppure eliminarne una con

```
unset nome
```

Per fare riferimento al valore di questa variabile dobbiamo far precedere il suo nome dal carattere "\$" ; cosi' se vogliamo sapere il valore della variabile "nome" dovremo scrivere:

```
echo $nome
```

Un "\" toglie il significato particolare a "\$". Per cui con

```
echo \$nome
```

si otterra'

```
$nome
```

Il comando "set" senza nessun argomento riporta l'elenco di tutte le variabili correntemente "settate" con il rispettivo valore. Una variabile puo' avere come valore anche una stringa costituita da piu' parole come in:

```
set nomi=(Mauro Massimo Alberto Carlo)
```

le parentesi sono necessarie in quanto

```
set nomi=Mauro Massimo Alberto Carlo
```

assegnerebbe "Mauro" alla variabile "nomi" e "setterebbe" le variabili "Massimo" "Alberto" e "Carlo". Queste parole saranno accessibili per mezzo di un indice:

```
echo $nomi[1] $nomi[4] $nomi[3]
```

ci dara' la prima la quarta e la terza parola della variabile "nomi":

```
Mauro Carlo Alberto
```

possiamo anche specificare le parole dalla prima alla terza:

```
echo $nomi[1-3]
```

oppure dalla seconda all'ultima:

```
echo $nomi[2-]
```

Tra parentesi quadre puo' essere presente o un numero, nel qual caso indica una determinata parola, o una coppia di numeri che indicano le parole tra quella indicata dal primo numero e quella indicata dal secondo comprese. Se e' assente il primo viene assunto essere "1", se e' assente il secondo viene assunto essere il numero di parole presenti nella variabile, questo numero e' sempre ottenibile come "\$#nome" dove "nome" e' il nome della variabile in questione. Come numero e' possibile specificare anche una variabile, ad esempio \$nome[\$i] se \$i vale 3 sara' Alberto. Per mezzo degli indici e' possibile cambiare il valore di una parola contenuta in una variabile. Es:

```
set nomi[3]=Pietro
```

sostituisce Pietro ad Alberto. E' possibile anche accodare delle parole ad una variabile o inserirne una in testa :


```
set nomi=($nomi Filippo)
```

se vogliamo accodare "Filippo" alla variabile "nomi".

```
set nomi=(Antonio $nomi)
```

se vogliamo mettere "Antonio" come prima parola della variabile "nomi".

Se e' facile inserire una parola in testa o in coda ad una variabile, l'unico modo possibile per eliminare una parola da una variabile e' quello di eliminare la prima parola per mezzo del comando "shift" : se scriviamo

```
shift nomi
echo $nomi
```

otterremo

Mauro Massimo Alberto Carlo Filippo

abbiamo eliminato cosi' il primo nome della lista. La forma \${nome} permette di isolare il nome di una variabile da eventuali altri caratteri che seguono. Ad esempio se "nome" vale "Fili", scrivendo

```
echo ${nome}ppo
```

otterrete "Filippo" mentre se aveste scritto:

```
echo $nomeppo
```

avreste ottenuto un bel "nomeppo: undefined variable". Usare una variabile che non e' ancora stata definita e' sempre un errore. Si puo' avere l'elenco delle variabili con i rispettivi valori per mezzo del comando "set". Alcune variabili hanno un significato particolare

##nome

Il numero di parole presenti nella variabile "nome"

\$?nome

Vale 1 se la variabile "nome" e' definita, altrimenti 0.

\$0

Il nome del file dal quale i comandi sono letti. Se il nome del file non e' conosciuto, fare riferimento a questa variabile causa un errore.

\${numero}

Corrisponde a \$argv[<numero>] con la differenza che anche se numero e' maggiore del numero di parole in argv cio' non causa errore.

ignoreeof

Se e' stata definita con "set ignoreeof", la shell ignora gli end-of-file dal terminale. Questo evita che una shell termini a causa di un <ctrl-D> battuto accidentalmente. Per scollegarsi sara' necessario digitare "logout" o "exit".

4.2.3. Comandi

Abbiamo detto che la cshell facilita al massimo l'uso dal terminale. Infatti la cshell fornisce oltre alla sostituzione di variabili che abbiamo gia' visto e che e' utile soprattutto ma non esclusivamente nella programmazione, anche altri due meccanismi di sostituzione: la sostituzione di storia ("history substitution") e l'alias. Se la variabile history e' stata definita allora la cshell ricordera' tutti i comandi impartiti permettendoci poi di richiamarli per mezzo del numero di comando oppure per mezzo dei primi caratteri che costituiscono il comando. Se il valore della variabile "history" e' un numero allora questo specifica il massimo numero linee di comandi che verranno "ricordati"; un limite ragionevole e' 100. Si puo' ottenere in ogni momento l'elenco dei comandi memorizzati tramite il comando "history". Quando in un comando e' presente il carattere "!", allora questo comando e' candidato alla sostituzione di storia: viene ricercato nella lista dei comandi uno che cominci con cio' che segue il punto esclamativo e sostituito alla parola o parte di parola che segue "!". Se subito dopo "!" c'e' un numero allora viene eseguito il comando specificato dal numero. Es:

```
!1      Il primo comando dato in questa sessione.
!v      L'ultimo comando che cominci per "v".
!!      Ripete l'ultimo comando eseguito.
!-2     Indica il penultimo comando eseguito.
```

Se nella variabile "prompt" e' presente "!" allora questo verra' sostituito con il numero del comando attuale. Es.:

```
set prompt='comando numero ! > '
```

il prompt sara' cosi':

```
comando numero 37 >
```

E' anche possibile selezionare particolari parole di un particolare comando. Prima si seleziona il comando nel modo

che avete visto, poi si seleziona la parola facendo seguire la selezione del comando da una serie di modificatori. Questi modificatori possono essere:

- :<numero>
Viene selezionata la parola specificata.
- :<numero>-<numero>
Le parole tra quella specificata dal primo numero e quella specificata dal secondo comprese. Le possibilità sono le stesse che per la specificazione delle parole nelle variabili.
- :\$
L'ultima parola. Se nella linea corrente e' gia' stato fatto un riferimento alla storia, "\$" indica l'ultima parola di quel comando, altrimenti indica l'ultima parola del comando precedente.
- :\$*
Tutte le parole tranne la prima (in pratica tutti gli argomenti del comando specificato). Per "\$*" vale lo stesso discorso che per "\$". Es:

```
vi !$
```

si entra in editor sull'ultima parola del comando precedente. Se abbiamo dato

```
cat pippo.p prova.p grafico.p | lpr
```

possiamo dare, ad esempio, vi !!:2 (del comando precedente (!!)) selezioniamo la seconda parola (:2)).

4.2.3.1. Correzioni di comandi

Un'altra possibilità offerta dalla cshell e' quella di correggere o modificare linee di comandi precedentemente impartiti. Una volta selezionato un comando e' possibile applicare delle correzioni in questo modo: supponiamo di avere dato un comando del genere:

```
pi pippo.p
```

possiamo richiamare l'editor vi sul file pippo.p scrivendo:

```
!pi:s/p/v/
```

Spiegazione:

Selezioniamo l'ultimo comando che inizia con "pi" (!pi) e lo modifichiamo sostituendo la prima "p" con una "v" (:s/p/v/); se il comando in questione era il comando precedente avremmo potuto scrivere ^p^v, che e' perfettamente equivalente a !!:s/p/v/. Al posto di "/" e' possibile usare qualsiasi altro carattere.

Oppure si puo' modificare un comando per mezzo di modificatori. I modificatori piu' usati sono:

- :t Significa "tail"; se l'ultima parola di un comando e' un path name allora viene sostituito dalla sua parte terminale ("/usr/paolo/testo" viene sostituito con "testo").
- :h Significa "head" : prende la parte iniziale di un path name ("/usr/paolo/testo" viene sostituito con "/usr/paolo").
- :r Una eventuale parte terminale del tipo "<stringa>" viene rimossa: ("capitolo.uno" viene sostituito con "capitolo")
- :p Stampa il comando senza nessun'altro effetto, e' molto utile soprattutto per verificare di aver selezionato il comando giusto.

Se vogliamo selezionare, ad esempio un comando che cominci con "sto" e appendere la stringa ".p" dobbiamo usare la forma !(sto).p perche' se avessimo scritto !sto.p avremmo selezionato un comando cominciante per "sto.p".

Esempi:

Supponiamo di avere dato i seguenti comandi:

```
3      vi testo.p
4      vi pippo.p
5      pi pippo.p
6      obj
7      pr testo.p pippo.p | lpr
```

Diamo ora i nostri comandi utilizzando la "history substitution". La prima linea di ogni comando e' cio' che e' stato digitato, la seconda l'espansione effettuata dalla cshell:

```
8 >    vi !:1
      vi testo.p
9 >    !!
      vi testo.p
```

```

10 >    !o >output
        obj > output
11 >    more !$
        more output
12 >    pr !$ ! lpe
        pr output ! lpe
13 >    ^e^r
        pr output ! lpr

```

Nel comando n. 12 abbiamo sbagliato, sostituendo "lpe" al corretto "lpr". Pur essendo linee di comandi piuttosto corte, la history substitution ci permette di ottimizzare il numero di caratteri battuti.

4.2.4. Alias

Un'altra facilitazione e' la possibilita' di definire delle abbreviazioni o sinonimi per comandi o per sequenze di comandi mediante il comando "alias".

```
alias l ls -la
```

definisce "l" come "ls -la", da ora in poi avremo a disposizione un nuovo comando il cui nome e' "l" equivalente in tutto e per tutto a "ls -la", una volta espanso il comando viene ricercato all'interno del comando risultante dei nuovi alias e viene ripetuta la ricerca fino ad un massimo di quindici volte dopo di che se c'e' ancora qualche alias non risolto viene emesso il messaggio "Alias loop.". A questo punto, per evitare sostituzioni infinite, il processo termina.

Nella stringa di definizione di un'alias e' possibile usare i comandi di sostituzione di storia e la linea di comando verra' usata come se fosse stato l'ultimo comando.

```
alias assegna 'set :2 = :1'
```

se provate "assegna 22 c" verra' trasformato in "set !:2 = !:1" e cioe' in "set c = 22".

E' comodo immettere i propri alias nel file di inizializzazione personale ".cshrc". Questi sono alcuni degli alias da noi usati:

```

alias a      alias
a          H  'history ! more'
a          h  'history ! tail'

```

```

a      L      '/usr/ucb/bin/l -Falspg * ! more'
a      P      '/bin/ps lax'
a      W      '/usr/ucb/bin/whereis
a      V      '/usr/ucb/bin/view
a      n      '/usr/bin/nroff -me'

```

4.2.5. La sostituzione di nomi di files

Se una parola contiene uno qualsiasi dei seguenti caratteri "[", "?", "*", "[" o comincia con il carattere "~" allora quella parola e' candidata ad essere sostituita con un nome di file. Questa parola e' vista come un modello e sostituita con una lista ordinata alfabeticamente dei nomi di file che corrispondono al modello. E' un errore se in una lista di parole che specifica una sostituzione di nome di file, nessuna corrisponde al nome di un file esistente, ma non e' un errore se questa corrispondenza esiste per almeno una parola. Soltanto per i metacaratteri "[", "?" e "*" questa regola e' valida, i caratteri "~" e "[" vanno visti come abbreviazioni. Il carattere "~" all'inizio di un nome di file e' usato come riferimento a direttrici : da solo viene trasformato nel valore della variabile home (per cui "echo ~" equivale a "echo \$home"), Quando e' seguito da un nome che consiste di lettere, cifre e "-", la shell cerca un utente con quel nome e sostituisce la sua direttrice (~rossi verra' espanso a "/usr/rossi" se questa e' la direttrice dell'utente rossi). Se il carattere "~" non e' all'inizio di una parola o non e' seguito da una lettera, una cifra o da "/" allora non viene interpretato in nessun modo.

La notazione "a(b,c,d)e" e' una abbreviazione per "abe ace ade". L'espansione dei nomi di files di una parola o di una variabile puo' essere inibita racchiudendola tra doppi apici (se \$nomefile vale "*" , echo \$nomefile dara' tutti i nomi dei files presenti nella direttrice corrente, mentre "echo "\$nomefile" dara' semplicemente *).

4.2.6. La sostituzione di comandi

Come la shell, anche la cshell permette la sostituzione di comandi: un comando o una serie di comandi racchiusa tra

"`" puo' essere messa ovunque avvenga sostituzione di comandi e la shell la rimpiazzerà con l'output prodotto da quei comandi:

```
set nome=`head -1`
```

ad esempio assegna a "nome" l'output del comando "head -1". E' questo l'equivalente di "read nome" della shell. head -n [file] legge le prime n linee da "file" se presente o da input, e le scrive su output. Tra "`" e' permessa la sostituzione di comandi e di variabili.

4.2.7. Redirezioni

Lo standard input e lo standard output di un comando possono essere rediretti con la seguente sintassi:

< nome

Apri il file "nome" (che e' prima soggetto a sostituzione di variabili, di comandi e di nome di file) come standard input.

<< parola

Legge l'input delle shell fino ad una linea consistente solo di "parola". Parola non e' soggetta a nessun tipo di sostituzione e ciascuna linea letta viene confrontata con parola prima di sottoporla a sostituzione di variabili e di comandi. Se "parola" e' racchiusa tra "'" o tra doppi apici o (nel caso sia costituita di un solo carattere) sia preceduta da "\" non viene effettuato nessun tipo di espansione. In caso contrario "\" puo' precedere "\$", "\", "`" per prevenire il loro significato particolare. Il testo risultante e' messo in un file temporaneo e fornito come standard input al comando.

> nome

>! nome

>& nome

>&! nome

Il file "nome" e' usato come standard output. Se il file non esiste allora e' creato; se il file esiste viene prima svuotato e il suo contenuto originale va perso.

Se la variabile "noclobber" e' stata definita, allora il file non deve esistere oppure essere un

file speciale : un terminale o il file vuoto per definizione (/dev/null). Questo e' molto utile per prevenire la distruzione accidentale del contenuto di un file. In questo caso il "!" sopprime questo controllo. La forma comprendente "&" ridirige sia lo standard output che lo standard error su file.

```
>> nome
>>& nome
>>! nome
>>!& nome
```

Usa "nome" come standard output come ">" solo che mette l'output alla fine del file. Se la variabile "noclobber" e' definita allora e' un errore se il file non esiste a meno che non si usi la forma "!". Per il resto e' simile a ">".

Se un comando e' eseguito in "background" (terminato da &) allora lo standard input di default e' il file vuoto "/dev/null". Se i comandi vengono letti da un file allora i files di input e di output sono quelli specificati dalle ridirezioni o dalla posizione del comando rispetto ad una pipe; solo nel caso sia presente "<<" l'input verra' letto dal file in cui si trovano i comandi stessi. I messaggi di errori possono essere rediretti attraverso una pipe usando "!&" invece di "!".

4.2.8. Espressioni

La cshell ha ereditato dal "C" anche alcune caratteristiche matematiche. Infatti la cshell puo' eseguire una serie di operazioni matematiche come addizione, sottrazione, moltiplicazione, divisione, modulo, or esclusivo, negazione, maggiore, minore, maggiore o uguale, minore o uguale. E' possibile l'uso di questi operatori sia direttamente sia all'interno di espressioni condizionali. Gli operandi di "uguale" e di "diverso" vengono considerati come stringhe, mentre tutti gli altri operatori operano su numeri. L'uso immediato di questi operatori e' permesso solo per assegnarne il risultato ad una variabile. La linea in cui questi operatori vengono usati deve presentare all'inizio il carattere "@". Vediamo subito qualche esempio esplicativo:

```
@ a = 8 + 2 * 3
```


viene assegnato alla variabile "a" (che non deve necessariamente essere stata prima definita) il risultato dell'espressione che segue. Da notare che ogni operatore deve costituire una singola parola. E' anche possibile usare le parentesi per modificare l'ordine di valutazione dell'espressione. Le precedenza degli operatori e', a partire dalla piu' bassa :

```

== e !=
    uguale e diverso

<= >= > e <
    minore o uguale, maggiore o uguale, maggiore,
    minore

+ e -
    * e /

```

Il risultato di questi confronti e' sempre una stringa che rappresenta un numero decimale. Sono disponibili come operatori anche alcune primitive di test su files. Queste primitive sono:

```

-f nome
    Ritorna "1" se nome e' un file normale.

-d nome
    Ritorna "1" se nome e' una direttrice.

-e nome
    Ritorna "1" se esiste un file di qualsiasi tipo
    con nome "nome".

-o nome
    Ritorna "1" se si e' i proprietari del file
    "nome".

-r nome
    Ritorna "1" se nome e' leggibile.

-w nome
    Ritorna "1" se nome e' scrivibile.

-x nome
    Ritorna "1" se nome e' eseguibile.

```

Per cui una espressione potrebbe essere fatta in questo modo:

```
@ a = ! -d miofile && -x miofile
```

Se "miofile" non e' una direttrice (! -d miofile) ed e' eseguibile (&& -x miofile) alla variabile a verra' assegnato il valore "1" altrimenti verra' assegnato "0".

L'uso di queste primitive e' pero' concepito per l'uso all'interno di espressioni condizionali e non e' molto utile al di fuori di essi.

4.2.9. Strutture di controllo

La shell include un certo numero di di comandi che possono essere usati per controllare la sequenza di esecuzione nei files di comandi.

4.2.9.1. Foreach

La sintassi di questo comando e' la seguente:

```
foreach nome ( lista di parole )
  <lista di comandi>
end
```

Spiegazione:

Ciascuna parola della lista di parole viene assegnata in successione alla variabile "nome" ed eseguita la sequenza di comandi compresa tra questo comando e il corrispondente "end".

Sia "foreach" che "end" devono comparire come unici comandi nella linea.

Il comando "continue" serve per passare all'esecuzione del ciclo successivo prima di aver eseguito tutti i comandi compresi nel ciclo.

Il comando "break" interrompe l'esecuzione del ciclo.

Quando questo comando e' letto da terminale la shell sollecita i comandi fino all'"end" con un "?". Se si commettono degli errori e' possibile interrompere l'immissione dei comandi premendo il tasto o <delete>.

4.2.9.2. If

Il comando "if" assoggetta l'esecuzione di comandi al risultato della valutazione di una espressione: un qualsiasi valore diverso da zero sta' per "vero"; zero sta' per falso. La sintassi e' questa:

```
if ( espressione ) comando
oppure if ( espressione ) then
  <lista di comandi> else if ( espressione2 ) then
  <lista di comandi> else
  <lista di comandi> endif
```

Spiegazione:

Nel primo caso se il valore dell'espressione e' diverso da zero allora viene eseguito il singolo comando che segue, altrimenti l'esecuzione procede al prossimo comando.

Nel secondo caso se il valore dell'espressione e' diverso da zero allora vengono eseguiti i comandi fino al primo "else", altrimenti se espressione2 e' verificata vengono eseguiti i comandi fino al secondo "else"; e cosi' via. Puo' essere presente un qualsiasi numero di coppie "else if" (anche nessuna) mentre e' necessario solo un "endif". Le parole "else" e "endif" devono essere le prime di una linea; "if" deve essere l'unico comando della linea in cui si trova oppure dopo un "else".

4.2.9.3. while

Esegue dei comandi finche' una qualsiasi condizione e' vera.

Sintassi:

```
while ( espressione )
  <lista di comandi>
end
```

Spiegazione:

Finche' "espressione" ha valore diverso da zero vengono eseguiti i comandi fino all'"end" corrispondente. Come per "foreach" il comando "continue" serve per passare all'esecuzione del ciclo successivo ignorando i comandi che seguono, e "break" interrompe l'esecuzione del ciclo. Se questo comando viene dato da terminale la shell sollecita l'introduzione dei comandi successivi con "?". Il comando "continue" serve per passare all'esecuzione del ciclo successivo prima di aver eseguito tutti i comandi compresi nel ciclo e il comando "break" interrompe l'esecuzione del ciclo piu' interno.

4.2.9.4. switch**Sintassi:**

```
switch ( parola )
case etichetta1:
    <lista di comandi>
    breaksw
case etichetta2:
    <lista di comandi>
    breaksw
default:
    <lista di comandi>
endsw
```

Spiegazione:

Dopo aver effettuato tutte le eventuali sostituzioni di variabili, di comandi e di nomi di files, "parola" viene successivamente confrontata con etichetta1, etichetta2 ..., che sono soggette alla sostituzione di variabile. L'esecuzione prosegue dopo l'etichetta che seleziona la parola. Etichetta1, etichetta2 ..., possono contenere i metacaratteri "?", "*" e "[...]" che mantengono il loro significato relativamente a "parola" e non ai nomi di files.

Se nessuna parola e' selezionata l'esecuzione prosegue dopo "default: "; se "default:" non esiste l'esecuzione prosegue dopo "endsw"; "breaksw" fa si che l'esecuzione prosegua dopo "endsw".

4.2.10. Un programma di esempio

In questo paragrafo tratteremo la realizzazione di una serie di comandi per la gestione di una rubrica telefonica.

4.2.10.1. I files di comandi

Se il primo carattere di un file contenente dei comandi e' "#" allora il contenuto di questo file verra' letto e interpretato dalla cshell. Il carattere "#" serve anche per introdurre commenti: la parte di linea che lo segue viene considerata un commento e quindi ignorata.

4.2.10.2. Ricerca di un nominativo: chenum

Vogliamo che il nostro comando accetti come parametri una lista di nomi o cognomi e ci riporti la o le linee che corrispondono alla richiesta. Questo si puo' fare semplicemente ricorrendo al comando grep per la ricerca ed a un "foreach" per scandire la lista di argomenti.

```
#
foreach nome ( $argv )
    grep $nome tele-elenco.
end
```

Si puo' aggiungere un controllo sull'esistenza dell'archivio e sul numero di argomenti:

```
#
if ( $#argv > 0 ) then
    if ( -f tele-elenco ) then
        foreach nome ( $argv )
            grep $nome tele-elenco
        end
    else echo "l'archivio non esiste"
    endif
endif
```

```

else
    echo "uso: chenum nomi"
endif

```

Spiegazione:

Se il numero di argomenti e' maggiore di zero (if (\$#argv > 0)), e se il file tele-elenco esiste e non e' una direttrice (if (-f tele-elenco), allora ciascun elemento della lista di argomenti viene cercato nell'archivio per mezzo del comando "grep", altrimenti nel caso non esista l'archivio viene emesso un messaggio di errore, nel caso non ci siano argomenti, il messaggio e' di spiegazione sull'uso del comando.

4.2.10.3. Inserimento di un nominativo: inserisci

L'inserzione di un nominativo si presenta apparentemente semplice, infatti la procedura piu' semplice potrebbe essere questa:

```

#
echo $* >>tele-elenco

```

Che pero' ha uno svantaggio : non effettua nessun tipo di controllo sul numero dei suoi argomenti, per cui potrebbe, al limite, essere chiamata senza nessun parametro. Il primo passo di affinamento puo' consistere nell'effettuare questo controllo assumendo che i parametri debbano essere almeno tre (nome cognome e numero di telefono):

```

#
if ( $#argv < 3 ) then
    echo Inserisci: numero di argomenti insufficiente.
else
    echo $argv >>tele-elenco
endif

```

A questo punto potremmo essere soddisfatti, ma ci manca ancora un tocco di raffinatezza: mantenere sempre ordinato il nostro archivio, in modo da renderlo piu' facilmente consultabile anche con mezzi diversi da quelli che abbiamo predisposto; aggiungiamo dunque il comando di sort alla nostra procedura:

```

#

```

```

if ( $#argv < 3 ) then
    echo Inserisci: numero di argomenti insufficiente
else
    echo $argv >> tele-elenco
    sort tele-elenco -o tele-elenco
endif

```

Spiegazione:

Se il numero degli argomenti e' minore di tre (if (\$#argv < 3) allora viene emesso un messaggio di errore, altrimenti si accoda all'archivio il nominativo desiderato e si ordina l'archivio (l'opzione "-o" di sort fa si che l'argomento successivo specifichi il nome del file su cui riversare il risultato dell'ordinamento. Non possiamo usare

```
sort tele-elenco > tele-elenco
```

perche' la shell svuoterebbe tele-elenco "prima" di lanciare l'esecuzione di "sort".

4.2.10.4. Rimozione di nominativi: rimuovi

Se dovessimo effettuare manualmente questa operazione entrerebbero in editor, ci posizioneremmo sulla linea desiderata e la cancelleremmo: ed e' proprio questo quello che faremo previa, naturalmente, la richiesta di una conferma:

```

#
if ( $#argv == 1 ) then
echo 'Verranno rimossi i seguenti nominativi:'
grep $1 tele-elenco
echo "Confermi la rimozione di $1 ?set risposta='head -1'
if ( $risposta == si ) then
    ed - tele-elenco<<fine
    g/$1/d
    wq
fine
else
    echo "Rimozione di $1 annullata"
endif
else echo "uso: rimuovi nominativo"
endif

```

Spiegazione:

Se il numero di argomenti e' uguale a uno allora vengono visualizzati i nominativi che verranno rimossi (grep \$1 tele-elenco), viene chiesta conferma della rimozione e letta una risposta (set risposta = 'head -1'), se la risposta e' "si" (if (\$risposta == si)), si richiama l'editor e vengono rimosse tutte le linee che contengano il nominativo specificato (g/\$1/d). Se la risposta e' diversa da si allora la rimozione viene annullata. Se il numero dei parametri e' diverso da uno ,viene emesso un messaggio esplicativo sull'uso del comando.

L'opzione "-" di "ed" evita che all'inizio e alla fine della sessione di editing venga visualizzato il numero dei caratteri che costituiscono il file. Da notare che la parola "fine" deve essere all'inizio della riga e che se dopo la fine della parola ci fosse anche un solo spazio, cio' avrebbe provocato un errore. Questo e' un errore piuttosto frequente e molti errori apparentemente inspiegabili sono dovuti a questo motivo.

4.2.10.5. Correzione degli errori

La fase di correzione degli errori, specialmente su procedure lunghe e complesse puo' essere molto laboriosa. Tre flags della cshell ci aiutano in questa fase molto delicata dello sviluppo di una procedura:

Con "-n" la shell controlla sintatticamente i comandi ma non li esegue, e' utile per trovare velocemente eventuali errori di sintassi; non e' questo comunque il tipo di errori piu' difficile da eliminare.

Con "-v" la variabile "verbose" viene definita. Quando questa variabile e' definita le linee di comando vengono visualizzate dopo la sostituzione di storia. Questa variabile puo' essere definita anche esplicitamente per controllare solo una parte di una procedura. E' utile per "scovare" sostituzioni di storia indesiderate.

Con "-x" la variabile "echo" viene definita. Quando questa variabile e' definita i comandi vengono visualizzati immediatamente prima dell'esecuzione, dopo che tutte le sostituzioni sono state effettuate. Questo e' lo strumento piu' potente nella fase di correzione degli errori in quanto

permette di avere una traccia del flusso di esecuzione di una procedura.

CAPITOLO 5

Strumenti software

5.1. AWK

Awk e' un potente linguaggio di programmazione la cui funzione e' di ricercare in un insieme di files delle parti che soddisfano certe condizioni e di effettuare sulle parti selezionate delle azioni specificate dall'utente. Awk, al contrario di grep, la cui unica azione e' di stampare le linee selezionate, permette di eseguire delle elaborazioni sulle parti selezionate o di modificare il formato in cui presentarle.

5.1.1. Uso di awk

Il comando

```
awk programma [files]
```

esegue i comandi di awk contenuti nella stringa "programma" sui files specificati, o, in assenza di essi, sullo standard input. Il programma di awk puo' essere messo in un file nel qual caso la linea di comando diventa

```
awk -f progfile [files]
```

Il programma in questo caso e' contenuto nel file "progfile".

5.1.2. Struttura di un programma

Un programma di awk e' una sequenza di istruzioni della forma

```
modello { azione }
modello { azione }
...
```

Ogni linea in ingresso viene confrontata con ciascun modello e per ogni linea che corrisponde al modello viene eseguita l'azione corrispondente.

Sia l'azione che il modello possono essere omessi ma non entrambe. Se non viene specificata nessuna azione per un modello la linea selezionata verra' semplicemente stampata. Se non c'e' un modello per una determinata azione allora quell'azione verra' effettuata su ogni linea in ingresso.

Poiche' sia i modelli che le azioni possono essere omessi, le azioni devono essere racchiuse tra parentesi graffe (tra "(" e ")") per distinguerle dai modelli.

5.1.3. Record e campi

Awk suddivide i suoi dati in "record" terminati da un separatore di record. Il separatore di record se non altrimenti specificato e' il carattere di <newline>, cosi' awk elabora il suo input una linea per volta. Il numero del record corrente e' disponibile nella variabile "NR".

Ogni record in ingresso e' considerato da awk come suddiviso in "campi". I campi sono normalmente separati da spazi, ma il separatore dei campi in ingresso puo' essere cambiato come vedremo. Ci si puo' riferire al primo campo di un record con \$1, al secondo con \$2, e cosi' via; \$0 e' l'intero record. E' possibile cambiare il valore delle variabili che contraddistinguono i campi. Il numero di campi

nel record attuale e' contenuto nella variabile "NF".

Le variabili "RS" e "FS" contengono rispettivamente il separatore di record e di campo, che possono essere cambiati semplicemente assegnando un nuovo valore a queste variabili, questo valore deve consistere di un qualsiasi singolo carattere. Il separatore di campo puo' essere definito nella linea di comando. Ad esempio

```
awk -F: -f progfile archivio
```

attiva awk con ":" come separatore di campo, il programma dovra' essere letto dal file "progfile" e i dati da elaborare sono nel file "archivio". D'ora in poi supporremo che il file "archivio" contenga queste linee:

```
Filippo:Bianchi:4538252:Via dei ciclamini 123:Milano
Romano:Filippetti:564381:Via dei turchi 13:Bologna
Woody:Allen:6738929:Largo manhattan 20:N.Y.:USA
Mauro:Rossi:223344:Via delle capinere 43:Roma
Carla:Bianchi:435627:Via mazzini 6:Foggia
Graziella:Bona:895568:Via 20 settembre 11:Milano
Marina:Rossi:3456712:Piazza verdi 14:Bologna
Sara:Svegliati:265496:Via primavera 21:Foggia
Franco:Pelati:938012:Piazza dante 3:Padova
David:Crosby:232563:4 way street:San diego,CA:USA
Antony:Freak:524637:Largo all'avanguardia 6:Bologna
```

5.1.4. Scrivere

Una azione puo' non avere un modello nel qual caso l'azione e' eseguita su tutte le linee. L'azione piu' semplice e' scrivere parte o tutto il record; Il programma di awk

```
{ print }
```

stampa tutti i record, copiando cosi' l'input sull'output senza alcuna modifica. Piu' utile e' stampare uno o piu' campi di un record. Per esempio

```
awk -F: '{ print $2,$1 }' archivio
```

scrive i primi due campi di ogni record in ordine inverso:

```
Bianchi Filippo
```

Filippetti Romano
 Allen Woody
 Rossi Mauro
 Bianchi Carla
 Bona Graziella
 Rossi Marina
 Svegliati Sara
 Pelati Franco
 Crosby David
 Freak Antony

In una istruzione "print" possono essere usate le variabili "NF" e "NR"; per esempio

```
awk -F: '{print NR,NF,$0}' archivio
```

scrive ogni record preceduto dal numero di record e dal numero di campi

```
1 5 Filippo:Bianchi:4538252:Via dei ciclamini 123:Milano
2 5 Romano:Filippetti:564381:Via dei turchi 13:Bologna
3 6 Woody:Allen:6738929:Largo manhattan 20:N.Y.:USA
4 5 Mauro:Rossi:223344:Via delle capinere 43:Roma
5 5 Carla:Bianchi:435627:Via mazzini 6:Foggia
6 5 Graziella:Bona:895568:Via 20 settembre 11:Milano
7 5 Marina:Rossi:3456712:Piazza verdi 14:Bologna
8 5 Sara:Svegliati:265496:Via primavera 21:Foggia
9 5 Franco:Pelati:938012:Piazza dante 3:Padova
10 6 David:Crosby:232563:4 way street:San diego,CA:USA
11 5 Antony:Freak:524637:Largo all'avanguardia 6:Bologna
```

E' anche possibile scrivere su di un file; il programma

```
{ print $1>"nomi" ; print $2 > "cognomi" }
```

scrive tutti i nomi (primo campo) sul file "nomi" e tutti i cognomi sul file "cognomi". Si puo' usare la notazione ">>" invece di ">" per accodare ad un file invece di crearlo. Il nome di un file puo' essere anche una variabile; per esempio

```
{print $1 > $2}
```

usa il contenuto del secondo campo come nome di file. Oltre al semplice "print" esiste l'istruzione "printf" che permette la scrittura secondo un certo formato. Il primo argomento di printf e' una stringa che specifica il formato con cui scrivere i rimanenti argomenti. La stringa di formato puo' contenere sia caratteri ordinari, che sono scritti inalterati, sia specificazioni di formato. Una specificazione di formato consiste del carattere %, un numero opzionale che specifica il minimo numero di caratteri di questo campo da scrivere, e un carattere che dice come

interpretare il campo da scrivere. Ci deve essere una specificazione di formato per ogni argomento di printf dopo il primo. Ad esempio

```
printf "%s", $1
```

scrive il primo campo come una stringa.

```
printf "%d
```

scrive il primo campo interpretandolo come un numero

```
printf
```

scrive il primo campo interpretandolo come una stringa, se il numero di caratteri in questo campo e' inferiore a dieci allora la stampa del campo viene preceduta da un numero di spazi sufficiente a raggiungere la lunghezza di dieci.

```
printf "%-10s", "1234"
```

Come nel caso precedente solo che la stampa del campo viene seguita da quella di un numero sufficiente a raggiungere dieci.

```
printf "%10.2f", "1234.5678"
```

Scrivo il primo argomento ("1234.5678") interpretandolo come numero con due cifre dopo il punto decimale in un campo minimo di 10 caratteri.

La sintassi di "printf" e' comunque identica a quella della corrispondente istruzione del linguaggio "C" al cui manuale si puo' fare riferimento per ulteriori chiarimenti.

5.2. Modelli

Un modello puo essere costituito da espressioni regolari (del tipo di "grep" o di "ed"), da relazioni aritmetiche o da combinazioni di queste.

5.2.1. I modelli BEGIN ed END

Il modello "BEGIN" indica l'inizio dell'input, cioè l'azione corrispondente a questo modello particolare viene eseguita prima che venga letto il primo record. Il modello "END" indica la fine dell'input, cioè l'azione corrispondente ad "END" verrà eseguita dopo che l'ultimo record è stato elaborato. Si può cambiare il separatore di campo a ":" prima di procedere all'elaborazione:

```
BEGIN { FS = ":" }
```

Oppure si possono contare i record in ingresso con

```
END { print NR }
```

Se "BEGIN" è presente allora deve essere il primo modello; "END", se usato deve essere l'ultimo.

5.2.2. Espressioni regolari

Le espressioni regolari riconosciute da awk sono equivalenti a quelle usate da "ed" e "grep". Ad esempio con

```
awk 'BEGIN {FS = ":" } /Roma/' archivio
```

otterremo

```
Romano:Filippetti:564381:Via dei turchi 13:Bologna
Mauro:Rossi:223344:Via delle capinere 43:Roma
```

cioè quei record che in qualche campo contengano la stringa "Roma"; In questo avremmo ottenuto lo stesso risultato con grep. Si può indicare anche che un modello deve selezionare un particolare campo in questo modo

```
$1 ~ /Roma/
```

In questo caso selezioneremo quei record che contengono nel primo campo la parola Roma, nel nostro caso

Romano:Filippetti:564381:Via dei turchi 13:Bologna

Si possono selezionare anche quei record il cui campo specificato non corrisponde al modello dato. Ad esempio

\$5 !~ /Milano!Bologna/

Seleziona quei record il cui quinto campo non contiene "Milano" o "Bologna", il carattere "~" sta per oppure. Applicando l'esempio al nostro file "archivio" otterremo

Woody:Allen:6738929:Largo manhattan 20:N.Y.:USA
 Mauro:Rossi:223344:Via delle capinere 43:Roma
 Carla:Bianchi:435627:Via mazzini 6:Foggia
 Sara:Svegliati:265496:Via primavera 21:Foggia
 Franco:Pelati:938012:Piazza dante 3:Padova
 David:Crosby:232563:4 way street:San diego,CA:USA

5.2.3. Espressioni relazionali

Un modello di awk puo' essere una espressione relazionale cioe' una espressione contenente degli operatori di relazione. Gli operatori disponibili sono:

>	maggiore
>=	maggiore o uguale
==	uguaglianza
!=	disuguaglianza
<=	minore o uguale
<	minore

ad esempio il modello

\$3 + 30 > \$5

seleziona quei record il cui terzo campo sommato a 30 e' maggiore del quinto campo. Se tutti gli operandi sono numerici allora i confronti vengono effettuati considerando tali operandi come numeri, altrimenti il confronto avviene tra stringhe.

5.2.4. Combinazioni di modelli

Un modello puo' essere costituito da una qualsiasi combinazione booleana di modelli usando gli operatori || (or), && (and) e "!"(not).

5.2.5. Intervalli di modelli

Il modello che seleziona una azione puo' anche costituito da due modelli separati da una virgola:

```
modello1,modello2 ( ... )
```

In questo caso l'azione e' eseguita per tutti i record tra quello selezionato dal primo modello ed un successivo selezionato dal secondo modello, incluso. Per esempio

```
NR == 10 , NR == 20 ( print )
```

scrive i record compresi tra il decimo e il ventesimo; e

```
awk -F: '$1 ~ /Mauro/, $1 ~ /Sara/' archivio
```

produce

```
Mauro:Rossi:223344:Via delle capinere 43:Roma
Carla:Verdi:435627:Via mazzini 6:Foggia
Graziella:Bona:895568:Via 20 settembre 11:Milano
Marina:Rossi:3456712:Piazza verdi 14:Bologna
Sara:Svegliati:265496:Via primavera 21:Foggia
```

5.3. Le azioni

Una azione in awk e' una sequenza di istruzioni terminate da una nuova linea o da un punto e virgola.

```
"" .          nr $1 length""
```

5.3.1. La funzione

La funzione `length` calcola la lunghezza di una stringa di caratteri oppure la lunghezza dell'intero record se usata senza parametri. Questo programma scrive ciascun record preceduto dalla sua lunghezza:

```
( print length , $0 )
```

questo programma poteva essere scritto anche così'

```
( print length($0) , $0 )
```

e applicato al nostro archivio produce

```
52 Filippo:Bianchi:4538252:Via dei ciclamini 123:Milano
50 Romano:Filippetti:564381:Via dei turchi 13:Bologna
47 Woody:Allen:6738929:Largo manhattan 20:N.Y.:USA
45 Mauro:Rossi:223344:Via delle capinere 43:Roma
41 Carla:Bianchi:435627:Via mazzini 6:Foggia
48 Graziella:Bona:895568:Via 20 settembre 11:Milano
45 Marina:Rossi:3456712:Piazza verdi 14:Bologna
45 Sara:Svegliati:265496:Via primavera 21:Foggia
42 Franco:Pelati:938012:Piazza dante 3:Padova
49 David:Crosby:232563:4 way street:San diego,CA:USA
51 Antony:Freak:524637:Largo all'avanguardia 6:Bologna
```

5.3.2. Variabili, espressioni e assegnamenti

In `awk` si possono usare delle variabili che vengono considerate numeriche o stringhe in dipendenza dal contesto: Per esempio, in

```
x = 1
```

`x` è chiaramente un numero, mentre in

```
x = "rosa"
```

è chiaramente una stringa. Le stringhe sono convertite in

numeri e viceversa qualora lo richieda il contesto. Per esempio

```
x = "3" + "4"
```

assegna 7 ad x. Stringhe che non possono essere interpretate come numeri in un contesto numerico hanno valore numerico zero

```
x = "rosa" + 22
```

assegna 22 ad x. Le variabili sono inizializzate alla stringa nulla che ha come valore numerico zero; questo elimina in molti casi l'inizializzazione delle variabili.

Tutte le operazioni aritmetiche sono effettuata in virgola mobile. Gli operatori aritmetici sono +, -, *, / e % (modulo). Esiste anche un operatore di incremento(++), e uno di decremento(--), che possono essere preposti o posposti ad una variabile per modificarne il valore, ad esempio

```
x++
```

equivale a

```
x = x + 1
```

Se gli operatori ++ o -- sono posposti allora il valore della variabile verra' modificata dopo essere stata usata; cosi'

```
y = x++
```

equivale a

```
y = x
x = x + 1
```

viceversa se gli operatori ++ o -- sono preposti il valore della variabile verra' modificato prima che sia utilizzata; cosi'

```
y = ++x
```

equivale a

```
x = x + 1
y = x
```

5.3.3. Le variabili che indicano campi

I campi in awk hanno tutte le proprietà delle variabili ed è possibile assegnare loro un valore. Ci si può riferire a dei campi con delle espressioni; questo programma ad esempio stampa gli ultimi due campi di ogni record

```
( print $(NF -1),$NF )
```

5.3.4. Concatenazione di stringhe

Delle stringhe possono essere concatenate semplicemente giustapponendole. Ad esempio

```
length( $1 $2 $3)
```

ritorna la lunghezza dei primi tre campi. Questo programma scrive il campo corrispondente al nome e quello relativo alla città di residenza per ogni record nel nostro archivio.

```
awk -F: '{print $1 " abita a " $5}' archivio
```

```
Filippo abita a Milano
Romano abita a Bologna
Woody abita a N.Y.
Mauro abita a Roma
Carla abita a Foggia
Graziella abita a Milano
Marina abita a Bologna
Sara abita a Foggia
Franco abita a Padova
David abita a San diego,CA
Antony abita a Bologna
```

5.3.5. Array

Gli elementi di un array non hanno bisogno di essere dichiarati, esistono solo per il fatto di essere stati menzionati, esattamente come per le variabili. Gli indici possono avere "qualsiasi" valore non nullo, incluse stringhe non numeriche. Come esempio di un convenzionale indice numerico, l'istruzione

```
x[ENR] = $0
```

assegna il corrente record all'elemento dell'array "x" il cui indice e' il numero di record corrente. In questo modo e' possibile in teoria elaborare tutto l'input in ordine casuale con un programma del tipo

```
{x[ENR] = $0}
END { ... programma ... }
```

La prima azione semplicemente memorizza ogni record in ingresso nell'array "x" e l'elaborazione vera e propria viene demandata alla fine della lettura di tutti i record.

Abbiamo detto che gli indici di un array possono non essere dei valori numerici; sfruttando questa capacita' possiamo scrivere un programma che conti il numero di residenti a Foggia e a Bologna nel nostro archivio

```
/Foggia/ { x["Foggia"]++ }
/Bologna/ { x["Bologna"]++ }
END      { print x["Foggia"] , x["Bologna"] }
```

5.3.6. Strutture di controllo

Awk dispone di potenti strutture di controllo: if-else, while, for e la possibilita' di raggruppare istruzioni. La sintassi di "if" e'

```
if ( espressione ) istruzione
```

oppure

```
if ( espressione ) istruzione
else istruzione
```

espressione e' una qualsiasi espressione valida per awk comprendente anche confronti di stringhe, selezioni mediante gli operatori "~", "!~", e gli operatori booleani "&&", "||" e "!", e naturalmente le parentesi per modificare l'ordine di valutazione delle espressioni. Una istruzione puo' essere composta o da una singola istruzione o da piu' istruzioni racchiuse tra parentesi graffe.
La sintassi di "while" e'

```
while ( espressione ) istruzione
```

Finche' l'espressione e' vera viene eseguita "istruzione". Questo programma scrive in ordine inverso i campi che costituiscono un record ognuno su una linea diversa

```
{
    i=NF
    while ( i > 0 ) {
        print $i
        --i
    }
}
```

La sintassi di "for" e'

```
for (espressione1; espressione2; espressione3) istruzione
```

ed e' perfettamente equivalente a

```
espressione1
while ( espressione2 ) {
    istruzione
    espressione3
}
```

Il seguente programma esegue lo stesso compito di quello presentato nell'esempio di while

```
{
    for ( i= NF ; i > 0 ; i--)
        print $i
}
```

Esiste un'altra forma per l'istruzione "for" che permette di scandire gli elementi di un array

```
for ( <variabile> in <array> )
    istruzione
```

che assegna successivamente a <variabile> gli elementi dell'array <array>. Il risultato non e' definito se si modifica il valore della variabile durante il ciclo oppure se si accede a qualche elemento dell'array diverso da quello corrente.

L'istruzione "continue" puo' essere usata per passare all'esecuzione del ciclo successivo durante l'esecuzione di "for" o di "while". L'istruzione "break" puo' essere usata per abbandonare prematuramente l'esecuzione di un ciclo. L'istruzione next fa si' che awk passi ad elaborare il record successivo. Infine l'istruzione "exit" fa si' che il programma di awk termini l'esecuzione.

5.4. YACC

I programmi per calcolatore hanno in genere bisogno di dati in ingresso e questi dati avranno una qualche struttura, semplice o complessa a seconda del tipo di programma in questione. Che si tratti del testo di un programma da compilare o di una serie di numeri da sommare, il programmatore si trova sempre ad affrontare il problema di riconoscere queste strutture per poi passare all'elaborazione dei dati vera e propria. Il riconoscimento della struttura dei dati in ingresso ad un programma ostacola, specialmente nel caso di strutture complesse, l'approntamento del programma vero e proprio. YACC si propone come aiuto nella descrizione delle strutture dei dati in ingresso. In pratica il programmatore deve fornire a YACC una descrizione della struttura dei dati, delle azioni da eseguire quando un certo costrutto e' stato riconosciuto, e una routine che legge le componenti elementari della struttura. Yacc generera' un programma scritto nel linguaggio "C" che esegue le azioni specificate quando una particolare struttura viene riconosciuta. Yacc e' stato usato per costruire compilatori "C" "APL", "pascal" e il linguaggio awk.

CAPITOLO 6

Il file system

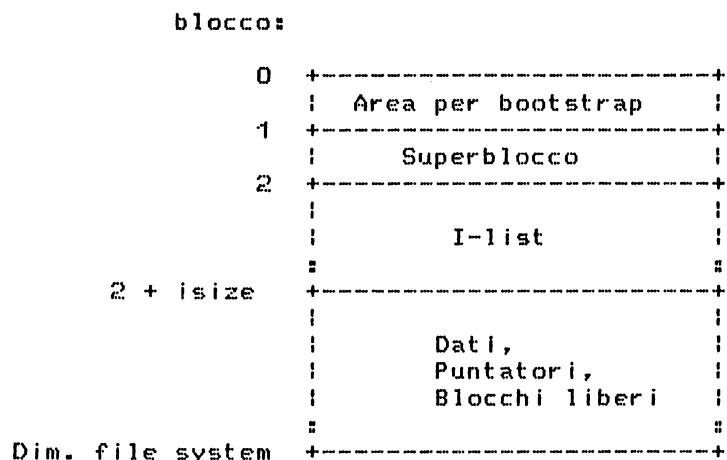
6.1. IL FILE SYSTEM

Nel sistema UNIX un file viene considerato come un array di bytes. Non ci sono altri tipi di strutture supportati dal sistema. I file sono raggruppati in una gerarchia di directory; una directory e' a sua volta un file in cui un utente non puo' scrivere.

Il file system risiede su un disco. Un disco viene visto come un array indirizzabile di blocchi da 512 bytes. Un file system si divide in quattro parti fondamentali.

6.1.1. Area di bootstrap

La procedura di bootstrap (in americano bootstrap e' l'accessorio per calzare gli stivali) serve per caricare in memoria centrale il nucleo (KERNEL) del sistema operativo, pertanto viene usata solo al momento di avvio. In genere tale fase avviene in almeno due tempi. All'inizio un codice rudimentale carica in memoria da disco un altro programma chiamato Initial Program Loader a cui viene dato il controllo; questo programma e' il responsabile del



caricamento in memoria del nucleo. Una volta lanciato il nucleo, non rimane traccia del secondo programma. L'IPL risiede sempre nel primo blocco (Block 0) del file system. Tale blocco e' chiamato "area di bootstrap" e non viene mai utilizzato dal sistema operativo.

6.1.2. Il superblocco

Il superblocco (blocco 1) contiene le informazioni vitali del file system. Senza il contenuto di questo blocco non e' possibile accedere a nessuna informazione contenuta in quel file system. Tra le informazioni piu' importanti ci sono : la dimensione del file system, la dimensione dell'area riservata alla i-list e quella dei blocchi disponibili.

La struttura del superblocco e' rappresentata in formato C come :

```
struct filesystem
{
    short    d_dimil;      /* Dim. i-list */
    long     d_dimvol;     /* Dim. volume */
    short    n_liberi;     /* Num. indirizzi in sliberi */
    long     d_liberi[50]; /* Lista blocchi liberi */
    short    d_numino;     /* Numero i-node in s_ino */
    int      d_ino[100];  /* Lista i-nodes liberi */
}
```

```

-
long      totbl_lib;    /* Tot. blocchi liberi */
int       totin_lib;    /* Tot. i-node liberi */
-
-
};

```

6.1.3. I-list

La i-list e' l'area che contiene tutte le informazioni relative ai files. Per ogni file viene occupata un'area di 64 bytes chiamata i-node. Gli i-nodes vengono identificati tramite un numero chiamato i-number; Tramite un i-number e' possibile identificare univocamente un file (all'interno di un file system). La struttura di un i-node e' la seguente :

```

struct diinode {
short      accessi;     /* Permessi e tipo di file */
short      numlink;     /* Numero di link */
short      user_id;     /* Codice del proprietario */
short      group_id;    /* Codice del gruppo */
long       dimensione; /* Dim. del file (bytes) */
char       indiriz[40]; /* Puntatori (13x3) */
long       taccesso;    /* Data ultimo accesso */
long       tmodifica;   /* Data ultima variazione */
long       tcreatz;     /* Data di creazione */
};

```

Ogni i-node contiene quindi informazioni relative al tipo di file (normale, speciale, directory); Il codice del proprietario e i bit di protezione; il numero di link (il meccanismo con cui UNIX permette di condividere un file tra piu' utenti); la grandezza del file; la data di ultimo accesso (in lettura), la data in cui e' stata fatta l'ultima modifica (scrittura); Infine ci sono i puntatori ai blocchi di cui si compone il file.

Ogni i-node contiene 13 puntatori; di cui i primi 10 vengono utilizzati come puntatori diretti ai blocchi del file. Se la grandezza del file supera i 5120 byte, viene utilizzato l'undicesimo puntatore. Tale puntatore anziche' indirizzare direttamente ad un blocco dati, punta ad un blocco contenente 128 puntatori diretti. In Questo caso si dice che il file utilizza un indirizzamento indiretto singolo

(single indirection). Se il file e' piu' grande di 70656 bytes, viene utilizzato anche il dodicesimo puntatore; Tale puntatore e' utilizzato per indirizzare un blocco contenente fino a 128 puntatori a blocchi contenenti a loro volta fino a 128 puntatori a blocchi dati; in questo caso si parla di doppio indirizzamento indiretto (double indirection). Infine se il file supera 8459264 bytes, viene usato l'ultimo puntatore. Questo puntatore indirizza un blocco contenente fino a 128 puntatori a blocchi di doppio indirizzamento indiretto ottenendo cosi' un indirizzamento triplo (triple indirection). Si puo' dedurre che la massima grandezza di un file e' di 1.082.201.087 bytes. La struttura e' visibile in Fig. 1

La gestione della i-list e' molto semplice. Nel superblocco c'e' un'area destinata a contenere i primi puntatori degli i-node disponibili (i-number). Quando questi puntatori finiscono il sistema operativo ricerca nella i-list i prossimi i-node disponibili fino al riempimento dell'area nel superblocco. Non esistono quindi strutture particolari.

6.1.4. Blocchi liberi

Lo spazio disponibile su disco e' gestito con una lista a puntatori (FIG. 2); ogni blocco della catena contiene un puntatore al successivo blocco e 50 puntatori a blocchi liberi. Lo scopo di questa organizzazione e' quello di ottimizzare le operazioni di I/O nella gestione dei blocchi liberi. Quando il sistema esaurisce i puntatori ai blocchi liberi contenuti nel superblocco, con una sola operazione di lettura riesce a riempire il superblocco e a ottenere l'indirizzo del prossimo blocco di puntatori. Al contrario, quando il superblocco e' pieno e ci sono altri puntatori da inserire nella catena, vengono scritti su un blocco 50 puntatori (in modo da liberare il superblocco) e questi puntatori vengono messi in testa alla catena; Quindi anche la fase di aggiornamento della catena comporta scarso I/O da parte del sistema operativo. Purtroppo questa efficienza si paga con l'impossibilita' di recuperare file erroneamente cancellati; infatti in un sistema attivo, le probabilita' che un blocco appartenente al file cancellato non venga utilizzato per altri scopi sono molto scarse; inoltre e' necessario che anche l'i-node relativo al file non venga riutilizzato. Allo scopo di mantenere una struttura piu' lineare possibile, quando un file viene rimosso i blocchi da lui occupati vengono rilasciati in ordine inverso rispetto a quello in cui era composto il file; questo per evitare di

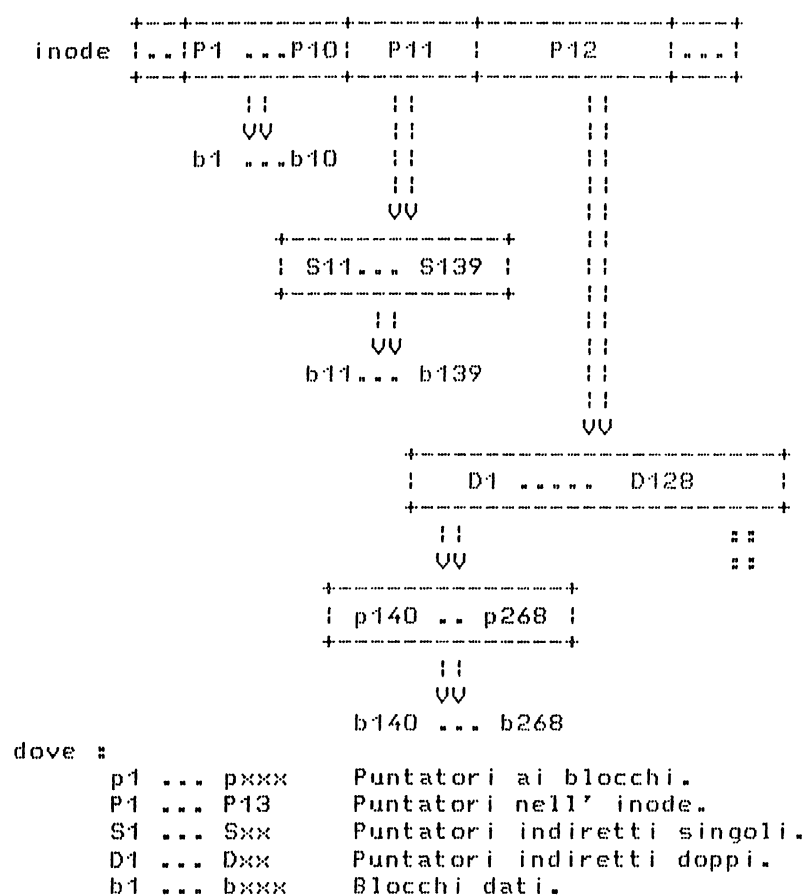


Fig. 1 Struttura del file.



Fig. 2 La catena dei blocchi liberi.

segmentare la catena dei blocchi liberi. Infatti se il file era composto da blocchi consecutivi, rimarranno tali anche nella catena dei blocchi liberi.

6.1.5. Directory

Strettamente legato al discorso del file system e' l'organizzazione delle directory; ogni directory e' composta da record di 16 bytes (in ogni blocco ci sono 32 records). UNIX inserisce nelle directory solo due informazioni: il numero di i-node (2 bytes) e il nome con cui il file e' conosciuto all'interno della directory (14 bytes). A questo punto e' abbastanza evidente il meccanismo dei link: basta avere due o piu' nomi (anche uguali se si tratta di directory diverse) con lo stesso i-number; Ogni volta che una directory referencia un file gia' esistente, il contatore di link viene incrementato, mentre ogni volta che viene rimossa una referencia questo contatore viene decrementato (Quando raggiunge il valore 0, il file viene rimosso).

Nell'i-node non sono contenuti ne' il nome (o i nomi) del relativo file ne' la posizione all'interno della struttura gerarchica del file sistem poiche' questo compito spetta alle directory. L'unico punto di riferimento sempre fisso e' la radice (root) del file system che corrisponde all'i-node numero 2; La posizione delle sottodirectory della radice non e' fissa ed e' ricavata dalle informazioni (i-number) contenute nella radice. Nella radice la directory ".." (padre) e' equivalente alla directory "." (FIG. 3). Quando in una directory il file viene rimosso, non viene cancellato il nome, ma soltanto azzerato l'i-number. Il record occupato verra' utilizzato per un altro file.

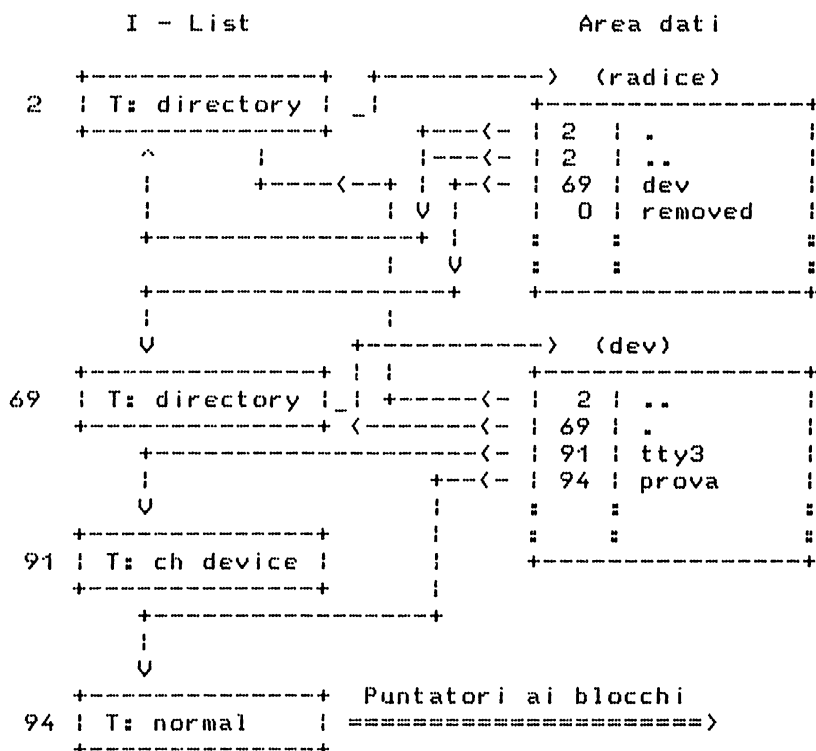


Fig. 3 File system e directory.

6.1.6. Organizzazione del file system

Poiche' un i-node definisce un file, l'organizzazione di tutto il file system e' basata sull'accesso all'i-node; per questa gestione UNIX memorizza una serie di tabelle non visibili all'utente. Il sistema operativo tiene in memoria una tabella di tutti gli i-nodes attivi (quelli corrispondenti a file aperti). Quando viene aperto un file, il corrispondente i-node viene messo in questa tabella e tutte le eventuali modifiche all'i-node vengono fatte in memoria. Nel momento in cui si e' fatto l'ultimo accesso al

file, l'i-node viene scritto nuovamente su disco e tolto dalla tabella. Tutte le operazioni di I/O vengono fatte attraverso la tabella degli i-nodes. Poiche' l'interfaccia con l'utente e' fatta con i pathnames e' necessario un algoritmo di trasformazione di pathnames in i-number. La trasformazione e' abbastanza semplice infatti partendo da un i-node conosciuto (es. la radice), viene ricercato all'interno della directory corrispondente il prossimo elemento del pathname; con questo otteniamo un i-number con il quale possiamo accedere all'i-node. Se il componente e' l'ultimo del pathname l'i-number che abbiamo trovato e' quello voluto, altrimenti il relativo i-node corrisponde alla directory dove deve essere cercato il prossimo componente.

Un processo utente puo' accedere al file system tramite alcune primitive di cui le piu' comuni sono :
Open, Create, Read, Write, Seek e Close.

6.1.7. Puntatori di I/O

In un sistema dove i file possono essere condivisi tra piu' utenti e processi, nascono delle problematiche relative alle operazioni di I/O; vediamo a grandi linee quali sono questi problemi e come sono stati risolti.

Nel segmento dati associato ad ogni utente, viene riservata un'area per una tabella dei file aperti (user open file table); Questa tabella e' costituita semplicemente da puntatori che consentono l'accesso alla tabella degli i-node; ad ogni file aperto viene associato un puntatore (I/O pointer) alla posizione del file dove dovra' essere eseguita la prossima istruzione di lettura o scrittura. In condizioni normali l'utente vede un file come una struttura sequenziale poiche' tale puntatore viene aggiornato automaticamente in funzione del numero di bytes letti nell'ultima operazione di lettura o scrittura eseguita. In caso di necessita' l'utente puo' ottenere un accesso random modificando il puntatore di I/O prima dell'operazione di lettura scrittura. Quando un file viene condiviso e' necessario: 1) consentire a processi parenti di utilizzare lo stesso puntatore di I/O; 2) avere un puntatore separato per ogni processo indipendente che deve accedere al file. E' evidente che il puntatore non puo' risiedere ne' nella tabella degli i-node ne' nella tabella dei file aperti per utente. Allo scopo di risolvere questo problema e' stata introdotta una nuova tabella (open file table), contenente i puntatori di I/O (FIG. 4).

I processi che condividono lo stesso file aperto (situazione che si verifica dopo una FORK) condividono anche

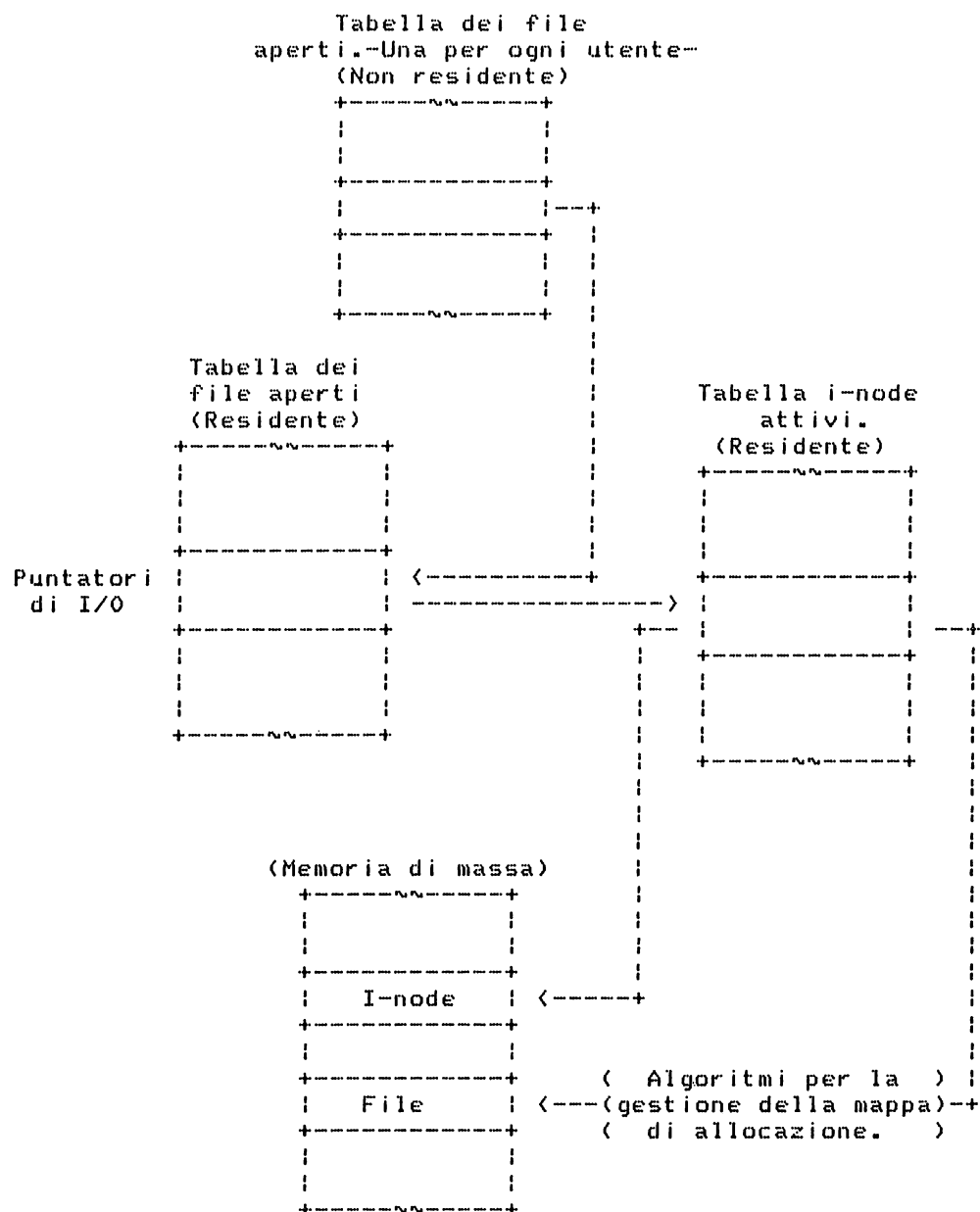


Fig. 4 Le tabelle per la gestione dei puntatori di I/O.

lo stesso puntatore nella open file table; mentre processi che eseguono open separate condividono solo l'i-node relativo al file e hanno un puntatore separato nella nuova tabella.

Vediamo le funzioni interne di alcune primitive:

Open	Converte un pathname in i-number e mette l'i-node relativo nella tabella degli i-node attivi. Nella tabella dei file aperti viene messo un puntatore all'i-node e infine nella user open table contenuta nel segmento dati dell'utente viene messo un puntatore alla tabella dei file aperti.
Create	Crea un i-node nella tabella degli i-node attivi e inserisce il relativo -number e nome nella directory; a questo punto esegue le stesse funzioni della open.
Read e Write	Accedono al file tramite la tabella degli i-node come descritto in precedenza.
Seek	Cambia il valore del puntatore di I/O.
Close	Rilascia le strutture generate dalle primitive Create e Open. Il sistema memorizza quanti processi condividono la i-node table e la open file table; Solo quando questi valori sono nulli le strutture vengono rilasciate.
Unlink	Decrementa il contatore di link in un i-node (che indica quante directory stanno utilizzando lo stesso i-node).

Quando il numero di processi che condividono un file e' nullo, se il contatore di link e' nullo, il file viene cancellato e la struttura rilasciata; In pratica la struttura associata ad un file aperto viene rilasciata solamente dopo che l'ultimo processo ha eseguito la Close. Questo metodo di "rilascio ritardato" evita di rimuovere la struttura del file prima che tutti i processi che lo stanno utilizzando abbiano eseguito la close. In altre parole una chiamata di unlink su un file che ha il contatore di link contenuto nell'i-node a 1, e che viene condiviso tra due processi ha l'effetto di rimuovere subito il nome del file dalla directory dove risiede, ma non distrugge la struttura del file fino alla chiusura dello stesso da parte di entrambe i processi. Grazie a questo meccanismo un file puo' essere rimosso mentre e' ancora aperto, in quanto verra' effettivamente cancellato (il nome ormai e' scomparso) dopo la chiusura. E' possibile utilizzare questa caratteristica

per avere file temporanei, non visibili e che vengono rimossi quando il processo termina (*). All'interno del sistema operativo questo metodo e' utilizzato per la gestione delle pipes.

6.1.8. Mount e unmount

Con UNIX e' possibile utilizzare contemporaneamente piu' unita' fisiche (device) ciascuna contenente un file system. Dopo l'operazione di bootstrap il sistema operativo utilizza un particolare file system designato come radice (root file system). Tale file system contiene tutti i files e programmi che permettono il corretto funzionamento di UNIX (Es. /etc/init, /bin/shell, /dev, etc...). A questo punto e' possibile estendere la struttura gerarchica di questo file system con il comando di mount; Il comando Mount serve per collegare ad una foglia (*) gia' esistente, la radice di quello nuovo. Supponiamo di avere una macchina con due unita' disco: una fissa e una amovibile (es. floppy) chiamata fd1; Supponiamo inoltre che il disco fisso sia designato come radice e che contenga la directory /usr2 vuota. Se ci spostiamo nella directory /usr2 (vedi comando cd) ed eseguiamo il comando ls, non avremo alcuna risposta poiche' la directory e' vuota. Se diamo l'istruzione di mount della unita' amovibile nella directory /usr2, abbiamo che il contenuto della directory diventa il contenuto della directory "/" del disco amovibile. In altre parole /usr2 non e' piu' una foglia ma un nodo dell'albero risultante (Vedi Fig. 5). L'utente che utilizza l'albero finale non si accorge che facendo riferimento alla directory /usr2 ed ai suoi figli, cambia unita' fisica. Al contrario l'istruzione di umount serve per "Sganciare" un file system (in precedenza montato) da un altro; Quindi per ripristinare la situazione originale basta dare il comando "/etc/umount /dev/fd1".

(*) Quando un processo termina, il sistema operativo chiude tutti i files eventualmente lasciati aperti. Il metodo di lasciare chiudere i files al sistema operativo, anche se non molto elegante, e' molto popolare tra i programmatori di sistema che cosi' facendo evitano (quando possibile) di gestire situazioni anomale (interrupt & C.).

* Per foglia si intende una directory vuota. Se il comando mount viene utilizzato su una directory contenente files o altre directory, non sara' possibile accedervi fino a quando non viene eseguita l'istruzione di umount.

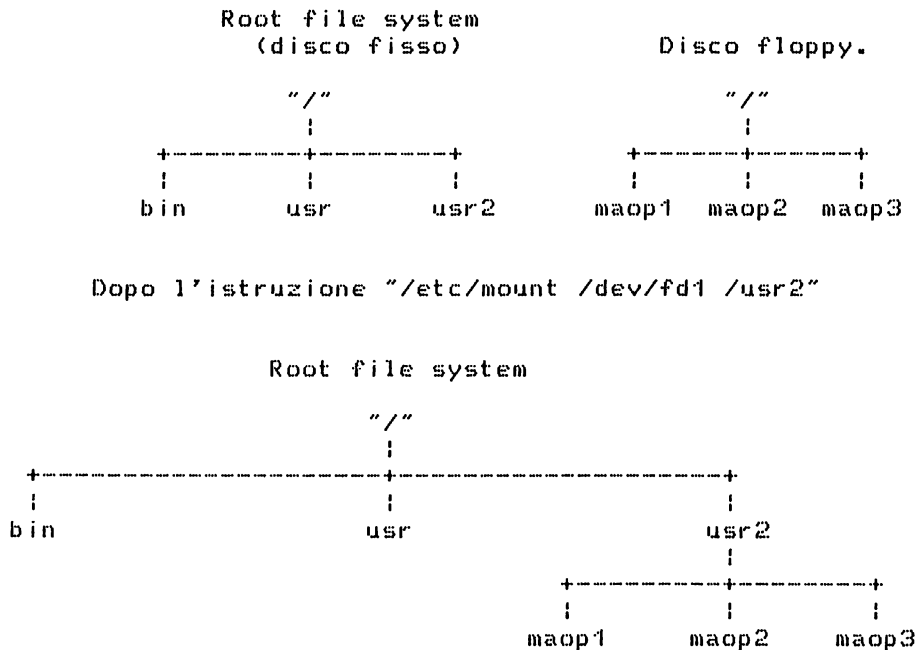


Fig. 5 Estensione del file system con il comando mount.

Vediamo ora come e' implementata l'istruzione di mount; il sistema operativo tiene una tabella dei file system montati contenente una coppia di dati formata dall'i-node e dal device montato. Durante la conversione da path-name a i-node, questa tabella viene consultata per vedere se l'i-node ottenuto vi e' contenuto; in caso affermativo tale i-node viene sostituito con quello della radice del file system residente sul nuovo device.

La gestione dei blocchi liberi viene effettuata sempre dal device in cui risiede il file creato. In questo modo non e' possibile avere aree comuni tra devices ma l'operazione di umount diventa estremamente semplice e veloce. Sempre per la stessa ragione, il sistema operativo non consente l'operazione di link tra file system diversi.

6.2. I FILES SPECIALI

In UNIX i files sono visti come un insieme di bytes accessibili anche in modo non sequenziale (random) nei quali vengono memorizzate e lette le informazioni. I files sono organizzati a loro volta in un insieme di blocchi su disco; ad ogni file corrisponde un insieme diverso di blocchi.

Esistono dei files speciali che a prima vista appaiono come quelli normali; infatti essi hanno gli stessi tipi di protezione e possono essere utilizzati nella maggior parte dei casi come i file tradizionali. C'e' pero' una differenza fondamentale: I files speciali corrispondono a unita' fisiche esterne (devices) come dischi o stampanti, oppure a segmenti di devices (ad esempio quando si divide un disco di grossa capacita' in piu' parti logiche), oppure a pseudo-device (come la memoria centrale).

All'interno del sistema operativo, i codici di gestione di questi files sono chiamati DEVICE DRIVERS e dove possibile, rendono i file speciali molto simili (se non uguali) ai file ordinari. la possibilita' di eseguire le chiamate di lettura (read) e scrittura (write) in quel file. Se il file interessato e' di tipo ordinario, le chiamate di lettura e scrittura comportano un trasferimento di informazioni da (o verso) l'insieme di blocchi su disco che costituisce il file. Se invece si tratta di un file speciale, le operazioni di lettura/scrittura causano il trasferimento dei dati direttamente da (o verso) il corrispondente device.

Supponiamo di eseguire le seguenti operazioni :

```
fd = open("prova",WRITE);
write (fd, "Messaggio di prova",18);
```

Otteniamo un file prova che contiene i caratteri : Messaggio di prova.

In altre parole abbiamo creato un struttura logica su disco formata da blocchi e contenente i caratteri voluti. Durante l'utilizzo ordinario di questo file non viene sentita la necessita' di studiare come e dove sono stati allocati nel disco i blocchi.

Prendiamo ora in considerazione il file speciale legato alla stampante chiamato lp; poiche' per convenzione tutti i files

speciali si trovano nella directory /dev il path-name completo della stampante e' /dev/lp.

Prendiamo in esame il seguente programma:

```
fd = open("/dev/lp",WRITE);
write (fd, "Messaggio di prova",10);
```

A differenza del caso caso precedente, non viene creato nessun file, ma viene stampato dalla stampante la scritta: Messaggio di prova.

Volendo si puo' eseguire la stessa operazione su un disco, con l'effetto di scrivere i dati su un blocco fisico del disco senza alcuna considerazione concernente la struttura logica del disco stesso; L'utilizzo sconsiderato di queste istruzioni apre la via alla corruzione di dati e in caso di suprema abilita' alla perdita di tutte le informazioni. Infatti supponiamo di eseguire l'ultimo programma su un file che rappresenta il disco e di essere posizionati sul superblocco (Sigh!); Nel momento in cui verra' eseguita l'istruzione di write, il vitale contenuto del superblocco verra' perduto e sostituito con i bytes relativi alla sequenza "Messaggio di prova", il sistema cadra' e non sara' possibile ripetere l'operazione di bootstrap.

L'unico rimedio e' quello di procedere al ripristino del disco in base all'ultima operazione di salvataggio dei dati. Per questa ragione quasi tutti i files speciali hanno i bit di protezione attivati.

Una differenza molto importante tra i files speciali e ordinari e' che con quelli speciali e' possibile fare alcune chiamate speciali al sistema operativo che con i files ordinari non avrebbero senso (ad esempio la chiamata stty permette di modificare la velocita' di trasmissione del terminale).

6.2.1. Tipi di files speciali

I files speciali possono essere di due tipi: a caratteri (character device) o a blocchi (block devices); In un file system sono riconoscibili da una lettera (c per quelli a caratteri e b per quelli a blocchi) che compare come primo carattere delle protezioni quando viene eseguito il comando ls -l. La differenza tra queste due caratteristiche e' di primaria importanza per operatori o programmatori di sistema; al contrario solo rare volte l'utente comune avra' bisogno di saperne di piu'.

I files speciali a blocchi (da ora in poi verranno chiamati

block devices) possiedono alcune particolari caratteristiche e sono trattati in modo particolare dal sistema. Un device a blocchi e' visto come una unita' accessibile anche non sequenzialmente (random) in blocchi di 512 bytes; in genere si tratta di unita' ad alta velocita' di trasferimento dati a cui e' associato un DMA (*). Il tipico esempio di block devices sono i dischi. Ci sono due differenze tra i block devices e i character devices: La prima e' che un file system puo' risiedere solo su un block device; la seconda e' che le operazioni di Input e Output con un block device vengono eseguite attraverso un buffer (**). Il kernel di UNIX e' fatto in modo da avere un buffer per tutti i blocchi di I/O, questo per mascherare all'utente la lunghezza fisica di un blocco (512 Bytes) e per fare apparire il device come un'unita' capace di gestire record di lunghezze diverse.

Purtroppo i buffers non sempre aumentano l'efficienza del sistema. Infatti se un programma esegue operazioni di lettura o scrittura con un numero elevato di bytes (512 o multipli), l'effetto del buffer e' solo quello di rallentare il trasferimento dei dati; Quando un processo esegue una richiesta di lettura di 512 bytes, il sistema prima trasferisce un blocco dall' unita' al buffer, poi trasferisce il blocco dal buffer all'area dati del processo. Questa situazione si verifica quando si effettuano grossi trasferimenti di dati (ad esempio durante il dump quotidiano (**)).

* I DMA (Direct Memory access) sono delle unita' hardware che permettono il trasferimento di dati dalla unita' periferica alla memoria ad alta velocita' senza l'intervento dell'unita' centrale. In genere vengono programmati dalla unita' centrale (CPU) che viene avvertita della fine del trasferimento con un segnale (Interrupt). Con questi componenti la CPU e' in grado di proseguire l'elaborazione mentre avviene il trasferimento.

** Un buffer e' un'area di memoria utilizzata per usi temporanei. Nel caso di I/O viene utilizzata per memorizzare il contenuto di un blocco (o piu') anche se il processo ne richiede alcuni bytes; Questo ha lo scopo di evitare una operazione sull'unita' nel caso di una lettura successiva. Inoltre nel caso in cui il processo esegua una operazione di scrittura, permette al processo di proseguire (scaricando i dati nel buffer) ritardando la scrittura effettiva ad un secondo tempo (ad esempio quando l'unita' non e' impegnata). La funzione del buffer e' apprezzabile maggiormente quando un blocco viene utilizzato frequentemente.

*** Il dump e' l'operazione di salvataggio dei dati su nastro magnetico.

I character devices sono i devices che non possono essere qualificati come block devices; inoltre non ricevono tutti i trattamenti particolari come i block devices, a seconda del tipo di character devices associato, il sistema riserverà alcuni trattamenti (o nessuno).

Ci sono alcuni character devices associati a unità disco e nastro che sono molto efficienti e potenti. Anziché creare uno o più buffer di 512 bytes, il sistema trasferisce con i DMA i dati direttamente dal disco all'area dati del processo che effettuato la richiesta. I vantaggi di questa situazione sono due: innanzitutto non viene sprecato tempo macchina per il trasferimento dal buffer all'area dati del processo (in quanto il buffer non c'è); Il secondo vantaggio deriva dal fatto che poiché non c'è il buffer, un processo è bloccato in memoria (lock) fino al termine dell'operazione di I/O (altrimenti i DMA non potrebbero trasferire i dati bloccando il device); In altre parole il processo può trasferire un numero elevato di bytes con una sola operazione di I/O.

Utilizzando i character devices è possibile trasferire in una volta sola una traccia, il che risulta senz'altro più efficiente che effettuare tale trasferimento con un blocco alla volta. una traccia un blocco alla volta. Inoltre eseguendo una operazione di I/O su nastro con un character device è possibile leggere un numero qualsiasi di bytes, mentre con un block devices si è costretti a leggere un record di 512 bytes.

Non tutti i character devices sono disponibili per questioni di efficienza, molti di essi esistono perché non è possibile qualificarli come block devices. Ad esempio è difficile utilizzare un terminale come block device con accesso random; altri tipi di unità che non risiedono nella categoria dei block devices sono stampanti, lettori di schede, perforatori di schede, lettori magnetici etc.

I block devices e i character non sempre appartengono a unità diverse. Ci sono alcuni casi in cui una unità può essere utilizzata tramite blocks o character devices a seconda delle esigenze. In genere si tratta di dischi e nastri; i block devices sono utilizzati per le operazioni di I/O degli utenti mentre i rispettivi character devices sono utilizzati per operazioni speciali o per assistenza sul file system. I character devices che corrispondono a block devices sono chiamati versione Raw (****) dei devices. Spesso guardando nella directory /dev si trova che i nomi dei character devices sono gli stessi dei block con la

 **** In inglese raw significa grezzo. In questo caso indica che i dati non vengono bufferizzati e vengono trasferiti direttamente nell'area dati del processo.

differenza che hanno una "r" anteposta (es. rp0 diventa rrp0).

6.2.2. Device primari e secondari

Un altro problema che viene incontrato nei file speciali e' quello di come sia possibile associare un device ad un file speciale.

Un file speciale contiene 3 informazioni : 1) Se e' un character o un block device; 2) Il numero primario. 3) Il numero secondario.

Un file speciale viene creato tramite l'istruzione:

```
mknod <file> <tipo> <n. primario> <n. second.>
```

L'uso di tale comando e' strettamente riservato all'operatore.

La differenza tra block e character device e' gia' stata discussa. Il numero primario indica al sistema operativo quale routine di controllo ha il compito di gestire il device. Il numero secondario permette di selezionare con precisione il device interessato. Ad esempio supponiamo di avere un sistema con 3 dischi fissi e 2 nastri; la tabella seguente mostra una ipotetica configurazione :

Devices	Num. primario	Num.secondario
-----	-----	-----
Disco 1	1	1
Disco 2	1	2
Disco 3	1	3
Nastro 1	2	1
Nastro 2	2	2

UNIX tiene in memoria centrale due tabelle: una per i character devices e l'altra per i block devices. Quando viene eseguita una chiamata al sistema operativo viene consultata la tabella appropriata, il numero primario e' utilizzato come indice in questa tabella in modo da localizzare la routine di controllo corrispondente; il numero secondario viene passato come argomento alla routine.

Un utilizzo del numero secondario puo' essere quello di dividere un disco in varie parti. Ad esempio in alcune utilizzazioni puo' essere utile avere separati il file system sistema e quello utenti in modo da fare il dump solo dei file utente. Quando si divide lo stesso supporto fisico

in piu' parti si parla di suddivisione logica (o Logical disks).

6.2.3. Protezione dei files speciali

Come abbiamo visto in precedenza l'utilizzo sconsiderato di un file speciale puo' portare anche alla distruzione del file system; allo scopo di evitare situazioni spiacevoli, molti files speciali hanno i bit relativi alle protezioni in lettura/scrittura attivati. Tali protezioni funzionano nella stessa maniera dei files ordinari (anche il bit s) pertanto non meritano particolare attenzione. Tuttavia e' utile vedere come in un caso particolare l'utente puo' modificare lo stato dei bit. Esiste un programma chiamato "write" (*) che consente di inviare un messaggio ad un utente. Il funzionamento di questo programma e' il seguente: 1) Viene cercato l'utente specificato in un file di sistema chiamato utmp (**) e risale al terminale su cui e' collegato. 2) apre il file del terminale (/dev/ttyx) in scrittura e ci scrive il messaggio "Message from user ttyx..." 3) apre il lettura lo standard input e lo copia nel file speciale relativo al terminale. Volendo e' possibile costruire il comando write in maniera semplice nel seguente modo: supponendo di essere nel tty4 e di voler mandare un messaggio a tty5 possiamo usare la linea di comandi:

```
echo Message from user tty4... >/dev/tty5; \
cat >/dev/tty5; \
echo EOT >/dev/tty5 \
```

Utilizzando tale sequenza otteniamo un risultato molto simile a quello della write. Un altro comando associato a write e' "msg" che serve ad abilitare o meno la ricezione dei messaggi. Il modo in cui opera "msg" e' banale; Se deve abilitare i messaggi, toglie la protezione in lettura e scrittura dal file speciale del terminale associato all'utente, altrimenti abilita il bit di protezione in scrittura. Supponendo di essere il terminale /dev/tty1 possiamo simulare il comando msg con:

Comando	Equivalente
-----	-----

* Vedi capitolo 2.

** /etc/utmp contiene gli id di tutti gli utenti collegati, il terminale e l'ora.

mesg y
mesg n

chmod go+rw
chmod go-rw

CAPITOLO 7

Rapide consultazioni

7.1. GLOSSARIO DEI TERMINI

albero (file system)

Gerarchia di files e directory. Ogni directory rappresenta un nodo dell'albero e puo' contenere sia files e directory. La radice dell'albero (root) rappresenta il punto di partenza da cui si espande l'intero file system.

argomento

La maggior parte dei comandi UNIX ricevono una lista di argomenti: Il comando

```
cat a b c d
```

consiste di un nome del comando chiamato "cat" e di 4 argomenti: "a", "b", "c", "d".

ascii

Standard di codificazione dei caratteri. Il sistema ascii usa 128 caratteri.

assembler

Linguaggio di programmazione a basso livello (piu' vicino alla macchina). Le istruzioni di questo linguaggio sono immediatamente comprensibili dalla cpu.

autoindentazione

Possibilita' fornita all'utente, attraverso un programma editor, di poter inserire righe di testo con il margine sinistro allineato a quello della linea precedente.

background

Un comando attivato senza aspettare la sua ultimazione e' chiamato comando in background. Viene utilizzato per esecuzioni che richiedono tempi medi o lunghi. Normalmente viene ridiretto sia l'input (se presente) che l'output (per non farlo interferire con il nostro lavoro). Un comando viene attivato in background con la "&" alla fine dell'istruzione.

```
nroff testo-n > testo-finale &
```

attiva il programma di nroff in background. L'output viene salvato nel file "testo-finale".

baud rate

Il baud (pronunciato "bod") e' una unita di misura della velocita di trasferimento dati e corrisponde ad un bit al secondo.

bit

La piu' piccola unita' di misura che definisce uno stato. Puo' assumere solo i due stati 0 o 1.

bit s

Substitute user id on execution. Presente talvolta in files di tipo eseguibile, permette all'utente che lo esegue di assumere temporaneamente le stesse caratteristiche e priorita' dell'utente proprietario del file.

bit t

Save swapped text after execution. Presente nei comandi di uso piu' frequente (vi, ls, more, cat, ...), salva l'immagine in memoria del comando dopo l'esecuzione per non doverlo ricaricare ad un ulteriore richiamo.

blank

Carattere bianco. Viene assunto come separatore di parole dalla shell e dai comandi UNIX.

blocco

E' il piu' piccolo numero di bytes (che non sia un singolo carattere) che puo' essere scritto o letto da una memoria di massa (dischi, nastri). La dimensione varia da dispositivo a dispositivo e da costruttore a costruttore, valori comuni per un blocco sono 256,

512, 1024 bytes. Su UNIX un blocco e' di 512 bytes.

buffer

Zona di memorizzazione temporanea nella quale vengono caricati dati in attesa di trasferimento. Viene usato per compensare le diverse velocita' di trasferimento e di elaborazione tra device fisici e logici. Video terminale, stampante, ecc... hanno un proprio buffer.

bug

Errore o imperfezione nel funzionamento di un programma.

byte

Unita' di misura composta da 8 bit. Un carattere e' rappresentabile all'interno di un byte. Un byte puo' avere $2^8=256$ stati diversi.

codice assoluto

Codice in formato binario interpretabile direttamente dalla cpu.

comando

Programma del sistema operativo. Un comando e' composto dal nome, eventuali opzioni ed una lista di argomenti (generalmente nomi di files) su cui il comando ha effetto.

compattare

Diminuire l'occupazione di memoria di massa di un file. Cio' e' reso possibile dall'istruzione "compact" che genera un codice corrispondente per poter riportare il file alle sue informazioni originarie.

compilatore

Programma che traduce un programma scritto in un linguaggio ad alto livello (C, Pascal, Lisp ...) in un codice in linguaggio macchina. Il risultato cosi' prodotto puo' essere eseguito dalla macchina.

console

Terminale privilegiato dal sistema su cui avviene la comunicazione tra operatore e macchina per l'amministrazione ed il mantenimento del sistema.

cpu

Central Processor Unit. Unita' centrale di un elaboratore; il suo funzionamento e' condizionato da sequenze di codici binari (programmi). E' in grado di eseguire operazioni elementari che, combinate assieme, permettono funzioni complesse. Ogni tipo di CPU interpreta codici binari (assembler) diversi.

cross reference

E' un programma che permette di ottenere un elenco dettagliato degli identificatori e del loro uso in un programma sorgente. E' utile nella fase di messa a punto dei programmi.

cshell

Interprete di comandi con sintassi delle strutture di controllo differenti dalla shell. Meccanismi di sostituzione di comandi utili all'utente.

current directory

Directory su cui sta avvenendo la comunicazione tra utente e macchina. E' possibile cambiare la current directory con l'istruzione "cd <nome-directory>". La current directory al momento in cui ci colleghiamo al sistema e' la home directory.

cursore

Trattino o quadratino luminoso del video terminale che ci comunica il punto in cui sta avvenendo l'input o l'output.

debugging

Attivita' di ricerca e correzione degli errori in un programma software. Viene fatta prima dell'ultimazione di un progetto software. Esistono programmi chiamati debugger che eseguono un programma passo a passo per facilitare il compito di debugging.

default

Le condizioni di default sono regole implicite che vanno in funzione se non sono state altrimenti specificate. Ad esempio, se non e' stato altrimenti specificato attraverso il comando "stty", <ctrl>S e' il carattere che ferma temporaneamente l'output che avviene su terminale.

device

Dispositivo periferico collegato al sistema. Dischi, nastri, terminali, stampanti sono devices. In UNIX i devices sono visti come files.

directory

Catalogo di files. Rappresenta un nodo nell'albero del file system.

directory figlia

Directory interna ad un'altra. La directory "sandra" di "/usr/utenti/sandra" e' figlia di "utenti".

directory foglia

Directory vuota. Non contiene ne' files ne' directory.

directory padre

Directory che ne contiene altre; queste ultime saranno sue figlie. Nell'esempio precedente, "usr" e' padre di "utenti".

dma

Direct memory access. Dispositivo hardware che permette il trasferimento di bytes tra le periferiche e la memoria centrale senza l'intervento della cpu che puo' cosi' proseguire le elaborazioni.

dump

Operazione di scarico dati. Dump e' l'operazione di copia su nastro magnetico delle informazioni presenti su disco. E' detto dump di memoria l'operazione di stampa o trasferimento su di un file del contenuto della memoria centrale o di una parte di essa.

editor

Programma che permette di creare, inserire, modificare e cancellare parti di un testo. Un editor puo' inoltre avere funzioni piu' complesse che sono di utilita' all'utente. Gli editor vengono utilizzati per stendere programmi, documenti, ecc...

end of file

Carattere che indica la fine dei dati. In UNIX corrisponde a <ctrl>D.

file

Sequenza di informazioni (record) memorizzati su una memoria di massa.

file speciali

In UNIX sono utilizzati per trattare i devices come files. Ad esempio, su un file speciale sono lecite le operazioni di open, read, write, seek e close. Queste operazioni si comportano diversamente in funzione del device a cui fanno riferimento.

file system

Struttura gerarchica composta da files e directory.

filtro

Programma che legge i suoi dati da standatd input e li immette, dopo aver eseguito qualche operazione su di essi, sullo standard output. Filtri molto usati sono grep e sort.

flag

Segnalatore. Variabile od altro che segnala una certa condizione.

formattare

Riferito ad un supporto di memorizzazione di massa indica l'operazione di registrare sul supporto stesso delle informazioni che verranno utilizzate poi dal dispositivo di controllo come segnali di riferimento per la regolazione della velocità di letture-scrittura e per la posizione dei dati.

Formattare un testo significa filtrare il testo attraverso un programma il cui output è il testo presentato attraverso una differente forma. Programmi di formattazione sono pr, nroff.

gruppo

UNIX considera gli utenti come suddivisi in gruppi, i componenti di un gruppo in genere sono utenti diversi che svolgono attività affini. Questa suddivisione viene usata soprattutto per regolare l'accesso ai files. Un utente può appartenere a più di un gruppo.

hardware

Termine che indica la parte "fisica" di un calcolatore in contrapposizione con "software", i programmi che ne governano il funzionamento.

home directory

Ogni utente ha associato una user-id (una parola che lo identifica nei confronti del sistema di addebito) e una "home directory" cui fa riferimento la shell nella ricerca dei comandi e dei files.

i-list

Area del file system riservata per memorizzare informazioni relative ai files. Una i-list è formata da più i-node. Non è possibile accedere direttamente a quest'area.

i-node

Blocco di 64 bytes contenente i dati relativi ad un file.

i-number

Numero per identificare un i-node.

interprete

Programma che è in grado di comprendere ed eseguire un codice. Alcuni linguaggi anziché essere trasformati in codice binario eseguibile dalla cpu, vengono interpretati. Ciò comporta una maggiore velocità nello sviluppo dei programmi a scapito di una minore velocità di esecuzione.

interrupt

Segnale hardware che può alterare la successione

logica di istruzioni che la cpu sta eseguendo. In UNIX sono implementati anche interrupt via software. Ad esempio, alcuni programmi terminano tramite il tasto <delete> o comandi del sistema operativo (kill).

kernel

Nucleo del sistema operativo che risiede sempre in memoria centrale. Gestisce le chiamate al sistema (exec, fork ...). Determina inoltre la ripartizione del tempo macchina tra i processi. Nella logica del sistema, il kernel viene identificato come processo numero zero. Appena avvenuto il caricamento, chiama in esecuzione il programma "/etc/init".

link

Meccanismo con cui UNIX permette di accedere allo stesso file con path name diversi.

login

Programma del sistema operativo che regola l'accesso al sistema. Codice simbolico in possesso di ogni utente.

macro

Istruzione il cui richiamo comporta l'esecuzione di una sequenza di istruzioni. Una macro puo' essere talvolta considerata come abbreviazione.

macro espansore

Programma che interpreta ed esegue le macro.

mark

L'azione di mark e' quella di contrassegnare una parte di testo per ottenere un successivo veloce ritorno.

memoria centrale

Memoria ad alta velocita' in cui vengono eseguiti i programmi. Tutti i dati contenuti in questa memoria vengono persi al termine di un programma.

memoria di massa

Memoria in cui risiedono permanentemente le informazioni (files). I dati memorizzati possono essere modificati o cancellati.

merge

Combinazione di due o piu' files in un'unico. Possono essere combinati secondo alcune regole, tipica e' quella di sort in cui piu' files ordinati vengono sommati in un'unico file anch'esso ordinato.

metacaratteri

Caratteri che assumono significato speciale. La shell,

gli editor vi e ed, grep e numerosi altri comandi interpretano ed espandono metacaratteri.

opzione

Parametro della forma "-(carattere)" che modifica il funzionamento di un programma.

password

Gli utenti si definiscono una parola chiave di sicurezza che ha lo scopo di fare accedere ai propri files.

path name

Nome di un file a partire dal livello piu' alto del file system: la root.

pattern

Espressione che incontra una o piu' parole che soddisfano certe caratteristiche.

periferica

Vedi device.

pipe

Meccanismo con cui UNIX permette il colloquio tra processi asincroni. Tramite la shell l'utente puo' utilizzare le pipe (con il simbolo "|") per connettere standard output e input di due programmi.

priorita'

Valore associato ad ogni processo che viene utilizzato dal sistema operativo per determinare la distribuzione del tempo macchina tra i processi.

processo

Esecuzione di un programma. Ogni processo e' identificato da un proprio numero.

programma sorgente

Programma scritto in un linguaggio comprensibile all'uomo. Necessita il supporto di un compilatore o di un interprete per essere eseguito.

prompt

Carattere o serie di caratteri che ci segnalano che la shell e' pronta ad accettare un nostro comando.

protezioni

Meccanismo con cui UNIX permette di controllare l'accesso ai files. L'utente puo' definire chi e come e' autorizzato ad utilizzare i propri files.

range

Intervallo.

reset

Segnale hardware che indica alla cpu di reinizializzare il sistema e le sue periferiche. In genere viene utilizzato per l'accensione della macchina e in caso di crash del sistema.

reverse

Differente modalita' di funzionamento del terminale per cui scritte bianche su sfondo nero vengono invertite in scritte nere su sfondo bianco.

ridirezioni

La shell offre la possibilita' di ridirigere l'input e l'output da e su altri files al posto dello standard input/output (il terminale video).

root

Riferimento iniziale dei path name del file system.

screen

Schermo. In genere viene utilizzato per indicare che un programma utilizza le funzioni del terminale video per una migliore facilita' d'uso, ad esempio uno screen editor.

shell

Interprete di comandi. Costituisce l'interfaccia tra utente e sistema operativo.

sintassi

Insieme di regole che determinano un linguaggio.

software

Programmi in genere, dal sistema operativo al programma utente.

spooler

Programma che si fa carico di stampare i files.

stampante

Device che consente di fare una copia su carta di programmi, dati, ecc..

standard error

Canale di comunicazione utilizzato per trasmettere eventuali messaggi di errore di un programma. Per default e' il terminale associato al processo. Puo' essere ridiretto.

standard input

Canale di comunicazione utilizzato da un processo per

ricevere eventuali dati dal terminale associato ad esso. Puo' essere ridiretto.

standard output

Canale di comunicazione utilizzato per trasmettere eventuali messaggi da parte di un processo. Per default e' il terminale associato al processo. Puo' essere ridiretto.

status

Ogni volta che un processo termina, ritorna un codice di terminazione che indica come il processo e' finito (0 = terminazione normale).

super user

Utente privilegiato addetto alla manutenzione ed alla gestione del sistema. Non esitono limiti di accesso od esecuzione.

swap

Area del file system utilizzata per la memorizzazione temporanea di processi che non possono restare in memoria centrale. Quando un sistema e' sovraccarico o dispone di una memoria centrale insufficiente, viene utilizzata a rotazione l'area di swap per consentire l'esecuzione di tutti i processi. Purtroppo l'utilizzo frequente dello swap comporta un aumento considerevole della inefficienza del sistema.

time sharing

Sistema in cui il tempo macchina viene diviso tra tutti i processi.

tty

Termine per indicare il terminale video.

utente

Persona che utilizza il sistema.

7.2. RIASSUNTO DEI COMANDI

7.2.1. Shell

sh

Interprete di comandi. Espansione di metacaratteri. Ridirezioni di input, output ed error da e su altri file. Connessione tra programmi attraverso pipe. Esecuzione di processi in background. Strutture di controllo: if - then - else, case, cicli di while e di for ... Programmazione in shell. Annuncia la presenza di posta. Personalizzazione di variabili e parametri. Files: .profile file di inizializzazione, .user inizializzazione personalizzata.

csh

Interprete di comandi. Caratteristiche della shell tutte presenti con sintassi a volte diversa. Definizione di alias di comandi. Sostituzione di comandi (history substitution). Variabili viste come arrays. Files: .login file di inizializzazione, .cshrc inizializzazione personalizzata.

vsh

Shell altamente interattiva. Lista della directory corrente sempre presente. Comandi composti per lo piu' da un unico carattere. Files: .vshrc comandi personalizzati.

7.2.2. Manipolazione files e directory

cat

Concatena uno o piu' files sullo standard output.

ccat

Concatena files compattati.

pr

Stampa files con intestazione ogni inizio pagina. Definizione della intestazione e della lunghezza della pagina. Stampa su piu' colonne. Stampa su colonne parallele di piu' files.

more

Filtra un file attraverso pagine. Proseguimento dell'output a seconda delle richieste dell'utente. Ricerche di patterns. Utilizzo di comandi shell.

compact

Rende il file in una forma compressa.

uncompact

Riporta il file alle condizioni precedenti al compact.

touch

Aggiorna la data di ultima modifica di un file o lo crea se non esiste.

lpr

Manda files sullo spool di stampa.

rm

Rimuove files. Se possiede piu' di un link, viene tolto solamente il link. Conferma interattiva. Rimozione di intere gerarchie di directory.

rmdir

Rimuove una directory solo se vuota.

cp

Fa una copia di un file su un'altro o di piu' files all'interno di una directory.

mv

Cambia nome ad un file. Sposta uno o piu' files da una directory ad un'altra.

cd

Cambia la directory di lavoro.

pwd

Riporta il path name della directory di lavoro

mkdir

Crea una nuova directory

od

Dump ottale, decimale o esadecimale di un file.

adb

Debugger interattivo. Puo' essere usato come editor per qualsiasi tipo di file (file binari, memoria).

head

Trasferisce in output la prima parte richiesta di un file.

tail

Trasferisce in output le ultime n linee o caratteri di un file. Trasferisce dall'ennesima linea o carattere in poi.

split

Spezza un grosso file in piu' files di n linee ciascuno.

strip

Toglie simboli esterni da un file oggetto.

prep

Riporta il contenuto di un file con una parola ogni linea.

ln

Crea un link ad un file.

test

Test per condizioni in shell. Confronto fra stringhe. Esistenza di files e loro caratteristiche. Operatori AND e OR fra condizioni.

ar

Mantiene archivi e librerie. Combina piu' files in uno. Creazione, lista, aggiornamento e cancellazione dei contenuti dell'archivio.

tee

Permette di passare l'output su piu' files e contemporaneamente sullo standard output.

7.2.3. accessi

login

Accesso alle risorse del sistema. Verifica la password, adatta il lavoro alle caratteristiche del terminale, attiva la shell.

passwd

Cambia o installa una password all'utente.

chmod Definisce i permessi di lettura, scrittura ed esecuzione per l'utente, il gruppo di utenti a cui appartiene e tutti gli altri.

umask Definisce i permessi al momento della creazione di un file.

chown Cambia la proprieta' di uno o piu' files.

chgrp Cambia il gruppo di appartenenza di uno o piu' files.

7.2.4. Documentazione

man Stampa il manuale del comando indicato

learn Lezioni di autoistruzione

apropos Spiega brevemente la funzione di un comando

7.2.5. Terminale

tty Stampa il nome del terminale

stty Definisce le modalita' di funzionamento del terminale.

reset Riporta le condizioni di default del terminale.

7.2.6. Comunicazioni

mail Spedisce un messaggio ad uno o piu' utenti. Legge la posta pervenuta

write Comunicazione diretta via terminale con un altro utente.

7.2.7. Situazione, processi e memoria

ps Informazioni sui processi attivi. Processi di nostra proprieta' e di tutti. Tempo, priorita', terminale su cui e' in esecuzione, status dei processi.

la Valori di carico del sistema.

wait Attende il termine dei processi in esecuzione.

kill Termina uno o piu' processi in esecuzione.

time Tempo di CPU, di sistema e reale di un comando.

nice Esegue un comando a bassa od alta priorita'

nohup Esegue un comando immune da segnali di interrupt da terminale.

who Riporta dove e da quando per ogni utente collegato al sistema.

date Data ed ora.

leave Comunica al sistema l'ora in cui ci si deve scollegare.

at Esegue comandi alla data indicata.

sleep Sospende l'esecuzione della shell per un certo tempo.

last Ultime login attivate.

set Tavola dei parametri e delle variabili allocate.

printenv Tavola dei parametri.

ls Lista il nome di uno o piu' files o directory. Protezioni, dimensioni, proprieta', data di accesso e modifica, links, i-number.

du Sommario dell'occupazione di memoria di una gerarchia di directory.

df Riporta l'ammontare dello spazio libero presente nei file system.

quot Statistiche, rispetto ad un file system, relative al numero di files ed occupazione complessiva di memoria di ogni utente.

7.2.8. Ricerche, filtri

sort Ordina o fa il merge di file ASCII linea per linea. Ordinamento alfabetico, numerico diretto o inverso. Ordinamento rispetto ad uno specifico campo o chiave. Distinzione o no tra maiuscole e minuscole. Soppressione di linee uguali tra loro.

tsort Ordinamento topologico. Ricava un ordine totale da un

ordine parziale.

cmp

Compara 2 files tra di loro

diff

Riporta le differenze tra due files. Linee modificate, linee in piu' o in meno. Produce comandi di editor per convertire un file su un'altro.

diff3

Compara 3 differenti versioni di un file riportando le parti di testo discordanti.

grep

Stampa le linee di un file che soddisfano un certo pattern. Stampa delle linee che non soddisfano il pattern. Numerazione delle linee incontrate.

egrep

Grep con possibilita' di espressioni regolari come pattern.

fgrep

Grep veloce. Lista di piu' patterns.

look

Trova linee in una lista ordinata.

wc

Conta linee parole e caratteri di un file.

uniq

Riporta le linee duplicate in un file.

find

Percorre il filesystem partendo da una directory cercando files che rispondono a certe caratteristiche. Ricerca per nome, proprietario, gruppo, permessi, tipo, links, dimensioni, i-number, date di accesso e modifica. Esecuzione di comandi sui file trovati. Combinazioni booleane di chiavi di ricerca.

file

Determina il tipo del file.

see

Mostra il contenuto di un file.

strings

Riporta le stringhe stampabili in un file di tipo binario.

awk

Linguaggio di ricerca patterns in archivi. Numero e lunghezza dei campi variabile. Ricerca per qualsiasi campo. Esegue operazioni sulle linee o campi incontrati. Formattazione dell'output. Dati visti come stringhe o come numeri. Strutture di controllo. Operatori aritmetici ed arrays.

sed

Versione di ed. Particolarmente utile per grossi file. Modifica il file a seconda dei comandi ricevuti. Strutture di controllo e di condizione.

7.2.9. Preparazione di documenti

ed

Editor di sistema. Accesso tramite comando ad ogni parte del file. Ricerche di patterns. Inserimenti, rimpiazzamenti, cancellazione e copie di qualsiasi parte del testo. Sostituzioni globali in una parte od in tutto il testo. Esecuzione di comandi shell.

vi

Visual editor. Posizionamento diretto del cursore sui caratteri del testo. Tutte le possibilita' di ed. Inserimento dell'output di un comando esterno all'interno del testo. Possibilita' di filtrare il testo attraverso un comando esterno.

nroff

Formattatore di testi. Istruzioni e testo in un unico file sorgente. Definizione dei formati della pagina, intestazione. Allineamento automatico dei margini. Centramento di righe. Indice automatico dei paragrafi. Note a pie' di pagina. Definizione di macro con argomenti.

troff

Formattatore per testi grafici e fotocomposizioni. Tutte le possibilita' di nroff. Controllo sulle serie di caratteri, loro dimensioni e posizione.

tbl

Produce tabelle di arbitraria complessita' compatibili con nroff e troff.

eqn, neqn

Composizione di formule matematiche. Piu' livelli di

parentesi, matrici, integrali, sommatorie ecc...

col

Adatta i reverse line feed per stampe con una passata sola per linea.

deroff

Toglie i comandi di troff dal file di input.

7.2.10. Linguaggi

cc

Compilatore del linguaggio C in quattro passi: macro espansione, controllo sintattico, traduzione in linguaggio assembler, ottimizzazione. Provvede al richiamo dell'assemblatore e del caricatore.

lint

Verificatore di programmi C. Cerca di fornire maggiori dettagli circa gli errori di sintassi presenti in un programma. Segnala anche probabili errori come variabili non usate o non inizializzate.

make

Usa le relazioni esplicitate nel file "makefile" per mantenere aggiornato un programma. Usa la data di ultima modifica di un file per decidere se un sorgente e' stato modificato dopo l'ultima compilazione. E' molto comodo per mantenere programmi molto complessi distribuiti su molti files e con regole di dipendenza complesse.

pas

Compilatore UCSD pascal. Genera un file in formato assoluto.

pcod

Traduttore UCSD pascal. Traduce in codice intermedio interpretato dal programma pxeq.

pi

Traduttore Berkeley Pascal. Traduce in un codice intermedio interpretato da px.

pxp

Stampa in bella forma un programma in Berkeley Pascal che deve essere corretto. Può stampare la traccia di esecuzione di un programma.

dc

Desk calculator. Calcolatore da tavolo programmabile. Esegue calcoli espressi in notazione polacca inversa usando tutta la precisione necessaria. Permette di specificare la base di uscita e di ingresso.

bc

Simile a dc, usa la normale notazione, la sintassi e' simile a quella del C.

as

Assemblatore.

7.2.11. Costruzione di compilatori

yacc

Costruttore di compilatori ed analizzatore sintattico (parser). Deve essere fornito di una descrizione della struttura dei dati e delle azioni da eseguire quando un certo costrutto e' stato riconosciuto. Yacc generera' un programma scritto nel linguaggio "C" che esegue le azioni specificate quando una particolare struttura viene riconosciuta.

lex

Generatore di analizzatori sintattici da usare poi con yacc.

7.3. COME E' STATO SCRITTO IL TESTO

Il testo e' stato scritto utilizzando il word processor "nroff" (vedi capitolo 2). I comandi usati sono quelli delle macro di "me". Oltre a quelli, sono state definite delle macro personali per meglio adattare le esigenze del testo finale al testo sorgente.

Gli argomenti che vengono passati alla macro vengono richiamati in maniera simile alle procedure di shell o cshell con \$1, \$2, ecc..., solamente che devono essere preceduti da "\\". Quando un argomento consta di piu' di una parola deve essere racchiuso tra doppi apici. I commenti si possono inserire facendoli precedere dai due caratteri \". Tutte le macro sono state definite usando caratteri maiuscoli per non coprire eventuali macro gia' definite da nroff.

Le macro istruzioni da noi definite sono le seguenti:

```
\" Macro che effettua un salto pagina
\" Se la pagina corrente e' pari.
.de PR
.if o .bp
..

\" Macro per le intestazioni dei capitoli
\" il primo argomento e' il numero del capitolo
\" il secondo il nome dello stesso.
.de CA
.hx          \" non scrive l'intestazione nella
              \" prima pagina del nuovo capitolo
.bp          \" va a nuova pagina
.oh '\"\\$2'%' \" intestazione pagine dispari
.eh '%UNIX manuale d'uso Cap. \\$1'
.ce          \" intestazione pagine pari
.ce          \" centra il numero del capitolo
CAPITOLO \\$1
.sp          \" lascia una riga bianca
.ce          \" centra il nome del capitolo
\\$2
.sp 8        \" lascia 8 righe bianche
..

\" Gestione dei paragrafi. Si richiama con
\" .PA numero-livello "nome paragrafo"
.de PA
.ne 8        \" se ci sono meno di 8 righe prima
```

```

\ " della fine della pagina va alla
\ " pagina successiva
.sp 3 \ " lascia 3 righe bianche
.sh \ $1 "\ $2" \ " chiama .sh con i due argomenti
.(x \ " mette il titolo nell'indice
\ $2
.)x
.sp 1 \ " lascia una riga bianca
.pp \ " indenta temporaneamente di 5 spazi
..

\ " Gestione figure: inizio figura. Le figure vengono
\ " trattate come un blocco da inserire o alla fine
\ " della pagina o all'inizio della prossima.
\ " IF Significa Inizio Figura; FF Fine Figura.

.de IF
.(z \ " Inizia Il Blocco
.hl \ " Genera una linea di Separazione
..

\ " Fine figura
\ " Questa macro assume come parametro la descrizione
\ " della figura per metterla
\ " in fondo insieme alla scritta "Fig.

.de FF
.ce \ " Centra il nome della figura
Fig. \ $1
.hl \ " Genera una linea di separazione
.)z \ " Chiude il blocco
..

\ " La seguente macro serve per centrare una singola
\ " linea, lasciando una linea bianca sopra e sotto.
\ " Si richiama con .CE "linea da centrare"
.de CE
.sp
.ce 1
\ $1
.sp
..

\ " Spiegazioni
\ " Inizio Spiegazione
.de (S
.sp
Spiegazione:
.in 5
..

\ " Fine Spiegazione
.de )S

```

```

.sp
.in 0
..

\" Macro per i termini del glossario
.de GL
.in 0          \" indentazione 0
.sp           \" lascia una riga bianca
\\$1          \" scrive l'argomento
.in 6         \" indenta di 6 il testo sottostante
..

```

Vediamo qualche esempio del testo sorgente contenente queste macro:

1) Questo e' l'inizio del 1o capitolo.

“nr” e' un comando di “me” che permette di gestire contatori. In particolare, viene comunicato che la variabile u viene inizializzata a 0 ed incrementata di uno ogni volta che viene richiamata. Questa variabile viene utilizzata per i numeri delle figure. Non bisogna così ricordarsi il numero della figura ogni volta che se ne immette una nuova, ma semplicemente incrementare il contatore “u”.

“nh” evita che le parole vengano spezzate per andare a capo, poiché, essendo nroff concepito per la lingua inglese, non sarebbero state rispettate le regole sintattiche.

“so” legge l'input dal file specificato, “.macros” contiene le macro appena viste.

“pp” indenta temporaneamente la riga sottostante di 5 spazi.

Le altre macro sono state definite da noi. Il numero che segue “.PA” indica il livello interno del paragrafo (livello 1 corrisponde ai capitoli, 2 ai paragrafi immediatamente interni, 3 ai sottoparagrafi e così via). Normalmente nroff espande gli spazi per esigenze di formattazione. Lo spazio assume così valore di delimitatore. I “\” all'interno delle parentesi tolgono il significato particolare degli spazi; non potranno così venire espansi. Ogni linea che inizi con punto viene interpretata come comando. Per ottenere un “\” in output e' necessario inserirne due nel testo sorgente (analogamente per ottenerne due sarà necessario inserirne quattro).

```

.nr u 0 1
.nh

```

```
..so /usr/SIST/fiorani/fior15/.macros
```

```
..CA 1 "Conoscere UNIX"
```

```
..PA 1 "Conoscere UNIX"
```

```
..PA 2 "IL FILE SYSTEM"
```

Questo capitolo tratta della struttura esterna del file system, rimandando la discussione su quella interna (\ la parte non visibile dall'utente \) all'ultima parte del libro. Il file system costituisce l'interfaccia tra l'utente e i dispositivi di I/O (\ input-output \). Qualsiasi entita' di UNIX e' vista come file, per cui tutto l'I/O e' indipendente dal dispositivo fisico in cui avviene ed e' trattato in maniera identica. I files del file system sono di tre differenti tipi : file normali, directory e file speciali.

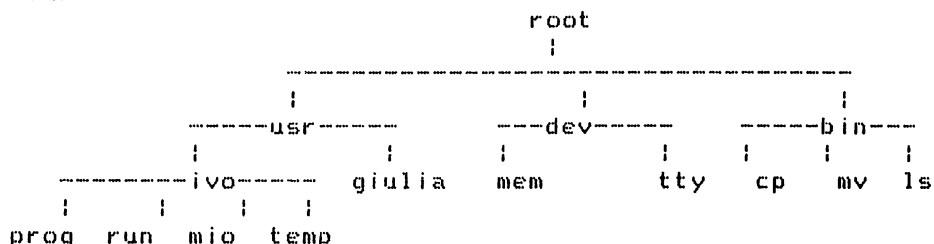
```
..PA 3 "File normali"
```

Un file normale e' strutturato come una sequenza di byte. UNIX organizza i files in blocchi di 512 byte. Un file di 513 caratteri (\ un carattere viene codificato in un byte \) occuperà così due blocchi di memoria. Questo però e' del tutto invisibile all'utente.

```
..PA 3 "Directory"
```

Una directory definisce il legame tra i nomi dei file ed i file stessi, e' praticamente un elenco di files che viene aggiornato automaticamente dal sistema a seconda delle richieste dell'utente. Ciascuna directory può contenere al proprio interno una o più directory (\ come normalmente avviene \). Il file system e' strutturato così in senso gerarchico, ad albero. La radice dell'albero e' la "root", una directory di sistema da cui dipende l'intero file system. Schematizzando, la struttura del file system e' visibile in Fig.\n+u.

```
..IF
```



```
..FF "\nu Schema dell'albero del file system."
```

La root e' rappresentata da uno slash (\ il carattere "/" \) e lo stesso slash e' usato per indicare i vari passaggi del path name di un file. Il path name e' la sequenza di directory da attraversare per giungere ad un determinato file.

```
..br
```

Per esempio, per riferirsi al file in figura 1

chiamato "mio", si parte dalla root e, separando con slash, si indicano le directory intermedie. Per cui,
 .CE "/usr/ivo/mio"
 e' il path name completo.

Analogamente per riferirsi al file "cp", si dara' il pathname /bin/cp. La figura e' chiaramente una semplificazione, poiche' normalmente in un sistema esistono migliaia di files.

.pp

In genere per riferirsi ad un particolare file

2) In questo stesso capitolo nel glossario dei termini viene utilizzata la macro ".GL":

.GL "argomento"

La maggior parte dei comandi UNIX ricevono una lista di argomenti: Il comando

.CE "cat a b c d"

consiste di un nome del comando chiamato "cat" e di 4 argomenti: "a", "b", "c", "d".

.GL "ascii"

Standard di codificazione dei caratteri.

Il sistema ascii usa 128 caratteri.

3) Sempre in questo capitolo, per la bibliografia viene utilizzata la macro ".np" che genera ed incrementa automaticamente i numeri.

.np

Brian W. Kernighan, "A Tutorial Introduction to the UNIX Text Editor",
 September 21, 1978, Bell Laboratories.

.np

Brian W. Kernighan, "Advanced Editing on UNIX",
 Bell Laboratories

7.4. BIBLIOGRAFIA

- (1) D.M. Ritchie and K. Thompson, "The UNIX Time-Sharing System", Comm. ACM, Vol. 17, No. 7, July 1974, pp.365-375.
- (2) Brian W. Kernighan, "UNIX For Beginners - Second edition", Bell Laboratories.
- (3) Brian W. Kernighan, "A Tutorial Introduction to the UNIX Text Editor", September 21, 1978, Bell Laboratories.
- (4) Brian W. Kernighan, "Advanced Editing on UNIX", Bell Laboratories
- (5) William Joy, "An Introduction to Display Editing with VI", December 3, 1979 (Revised for version 3.1), University of California, Berkeley.
- (6) S. R. Bourne, "An Introduction to the UNIX Shell", November 12, 1978 , Bell Laboratories.
- (7) William Joy, Susan L. Graham, Charles B. Haley, "Berkeley Pascal User Manual Version 1.1 - April, 1979", University of California, Berkeley.
- (8) Brian W. Kernighan, "Programming in C - A Tutorial", Bell Laboratories.
- (9) Dennis M. Ritchie, "The C Programming Language - Reference Manual", Bell Laboratories.
- (10) Brian W. Kernighan, "UNIX Programming - Second Edition", Bell Laboratories.
- (11) Gautier, Richard L., "Using the UNIX system.", Reston Publishing Company, Inc., Prentice-Hall.
- (12) Eric P. Allman, "Writing Papers with Nroff using -me", University of California, Berkeley.
- (13) K. Thompson, "UNIX Implementation", Bell Laboratories.
- (14) William Joy, "An Introduction to the C shell", December 20, 1979 (Revised for the third Berkeley Distribution), University of California, Berkeley.

- (15) Kurt Shoens, "Mail Reference Manual", Version 1.3, December 2, 1979 Bell Laboratories.
- (16) William Joy, "Ex Reference Manual Version 2.0 - April, 1979" April 6, 1979, University of California, Berkeley.
- (17) Robert Morris, Ken Thompson, "Password Security: A Case History", April 3, 1978 Bell Laboratories.
- (18) D. A. Nowitz M.E. Lesk, "A Dial-Up Network of UNIX Systems", August 18, 1978 Bell Laboratories.
- (19) D. A. Nowitz, "Uucp Implementation Description", October 31, 1978 Bell Laboratories.
- (20) J. F. Maranzano S. R. Bourne, "A Tutorial Introduction to ADB", May 5, 1977 Bell Laboratories.
- (21) Brian W. Kernighan, "RATFOR - A Preprocessor for a Rational Fortran" Bell Laboratories.
- (22) Lorinda Cherry, Robert Morris, "BC - An Arbitrary Precision Desk-Calculator Language", November 12, 1978 Bell Laboratories.
- (23) Robert Morris, Lorinda Cherry, "DC - An Interactive Desk Calculator", November 15, 1978 Bell Laboratories.
- (24) Brian W. Kernighan, "The M4 Macro Processor", July 1, 1977 Bell Laboratories.
- (25) Stephen C. Johnson, "YACC: Yet Another Compiler-Compiler", July 31, 1978 Bell Laboratories.
- (26) M. E. Lesk and E. Schmidt, "LEX: A Lexical Analyzer Generator", December 1980 Bell Laboratories.
- (27) Alfred V. Aho, Brian W. Kernighan, Peter J. Weinberger, "AWK - A Pattern Scanning and Processing Language (Second Edition)", September 1, 1978 Bell Laboratories.
- (28) Lee E. McMahon, "SED - A non Interactive Text Editor", August 15, 1978 Bell Laboratories.
- (29) Kenneth C. R. C. Arnold, "Screen Updating and Cursor Movement Optimization: A Library Package", University of California, Berkeley.
- (30) J. A. Hawley and W. B. Meyer, "MUNIX, A Multiprocessing Version of UNIX", M.S. Thesis, Naval Postgraduate School, Monterey, Cal. (1975).

- (31) W. Teitelman, "INTERLISP Reference Manual", Xerox Corp. Palo Alto Research Center, Palo Alto, Calif., Dec 1978.
- (32) P. Brinch Hansen, "Structured multiprogramming", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978.
- (33) A. Celentano A. Tecchio, "Il Sistema Operativo UNIX", prima edizione Maggio 1982, Cooperativa Libreria Universitaria del Politecnico - Milano.

*Serie
Linguaggi e
strumenti di programmazione*

**Ivo Quartirolì
Massimo Fusaro
Stefano Smareglia**

**UNIX
Introduzione al
sistema operativo**

Il sistema operativo Unix (marchio registrato Bell Laboratories) è in rapida e universale espansione nel settore dei mini e microelaboratori.

Il volume che presentiamo colma una vistosa lacuna editoriale: sull'argomento non esiste infatti alcuna documentazione in lingua italiana. Il testo si presenta chiaro e leggibile anche allo studente al suo primo approccio con la programmazione, ma riesce a soddisfare anche l'utente più sofisticato.

La trattazione, corredata da numerosi esempi, passa attraverso diversi livelli:

- introduzione alla "filosofia" di Unix;*
- confidenza con i primi comandi;*
- spiegazione dettagliata degli editor di sistema, compilatori, istruzioni;*
- uso dei linguaggi di controllo: shell e c-shell;*
- struttura interna del sistema operativo e suo uso avanzato.*

Compendia il tutto una parte finale di rapida consultazione, con il riassunto dei comandi e un utile glossario dei termini.

Quartiroli Fusaro Smareglia **UNIX**

clup