Note sul
Sistema Operativo AIX

G. Vitillaro

Aprile 1991

IBM

Contents

Breve storia di UNIX

- Nasce nei Bell Laboratories nel 1969 - Ken Thompson e Dennis Ritchie da Multics.

  - Il progetto MULTICS per un sistema operativo time sharing multi-utente termina senza un buon risultato nel 1969.
  - Il gruppo dei Bell Laboratories si trova temporaneamente senza un sistema di calcolo interattivo.
  - Thompson e Ritchie disegnano un nuovo File System che si evolvera' in una delle prime versioni del File System di UNIX.
  - Thompson scrive un programma che simula il File System proposto e codifica un Kernel preliminare per un GE 645.
  - Allo stesso tempo scrive un gioco, "Space Travel", in Fortran per il sistema GECOS (Honeywell 635).
  - Non soddisfatto ricodifica il programma per il PDP-7.

- ♦ Thompson diviene cosi' padrone del PDP-7, ma il suo ambiente di lavoro richiede un cross-assembly tra il GECOS e il PDP-7
- ♦ Per creare un migliore ambiente di sviluppo Thompson e Ritchie implementano il loro disegno di Sistema Operativo sul PDP-7, che include una versione preliminare del File System, il gestore dei processi e un piccolo insieme di utilita'.
- ♦ A questo punto il nuovo sistema diviene autonomo e non ha piu' necessita' del GECOS come ambiente di sviluppo.
- ♦ Al nuovo sistema viene dato il nome di UNIX in contrapposizione al nome MULTICS coniato da Brian Kernighan.
- ♦ UNIX e' scritto da programmatori per programmatori.

- D.M. Ritchie scrive un compilatore per il linguaggio C e nel 1973 Ritchie e Thompson riscrivono il Kernel di UNIX in C.

- ♦ Thompson diviene cosi' padrone del PDP-7, ma il suo ambiente di lavoro richiede un cross-assembly tra il GECOS e il PDP-7
- ♦ Per creare un migliore ambiente di sviluppo Thompson e Ritchie implementano il loro disegno di Sistema Operativo sul PDP-7, che include una versione preliminare del File System, il gestore dei processi e un piccolo insieme di utilita'.
- ♦ A questo punto il nuovo sistema diviene autonomo e non ha piu' necessita' del GECOS come ambiente di sviluppo.
- ♦ Al nuovo sistema viene dato il nome di UNIX in contrapposizione al nome MULTICS coniato da Brian Kernighan.
- ♦ UNIX e' scritto da programmatori per programmatori.

- D.M. Ritchie scrive un compilatore per il linguaggio C e nel 1973 Ritchie e Thompson riscrivono il Kernel di UNIX in C.

- ♦ Per la prima volta un Sistema Operativo viene scritto in un linguaggio differente dall'assembler della macchina ospite.
- ♦ Cio' permettera' di "portare" il nuovo sistema su una grande varieta' di piattaforme hardware.

- Nel 1974 viene dato in licenza alle Universita' per scopi didattici.

  - ♦ Il source viene distribuito insieme al sistema.
  - ♦ Cio' permette un facile adattamento a particolari richieste.
  - ♦ Nel 1977 il numero di sistemi UNIX installati raggiunge le 500 unita' delle quali 125 in Universita'.

- Fino al 1980 UNIX rimane confinato nell'ambiente delle Universita' Americane, dei laboratori di ricerca collegati con la rete ARPA del Dipartimento della Difesa Americano.

- Nel periodo tra il 1978 e il 1982 i Bell Laboratories combinano diverse versioni in un singolo sistema, conosciuto come UNIX System III.

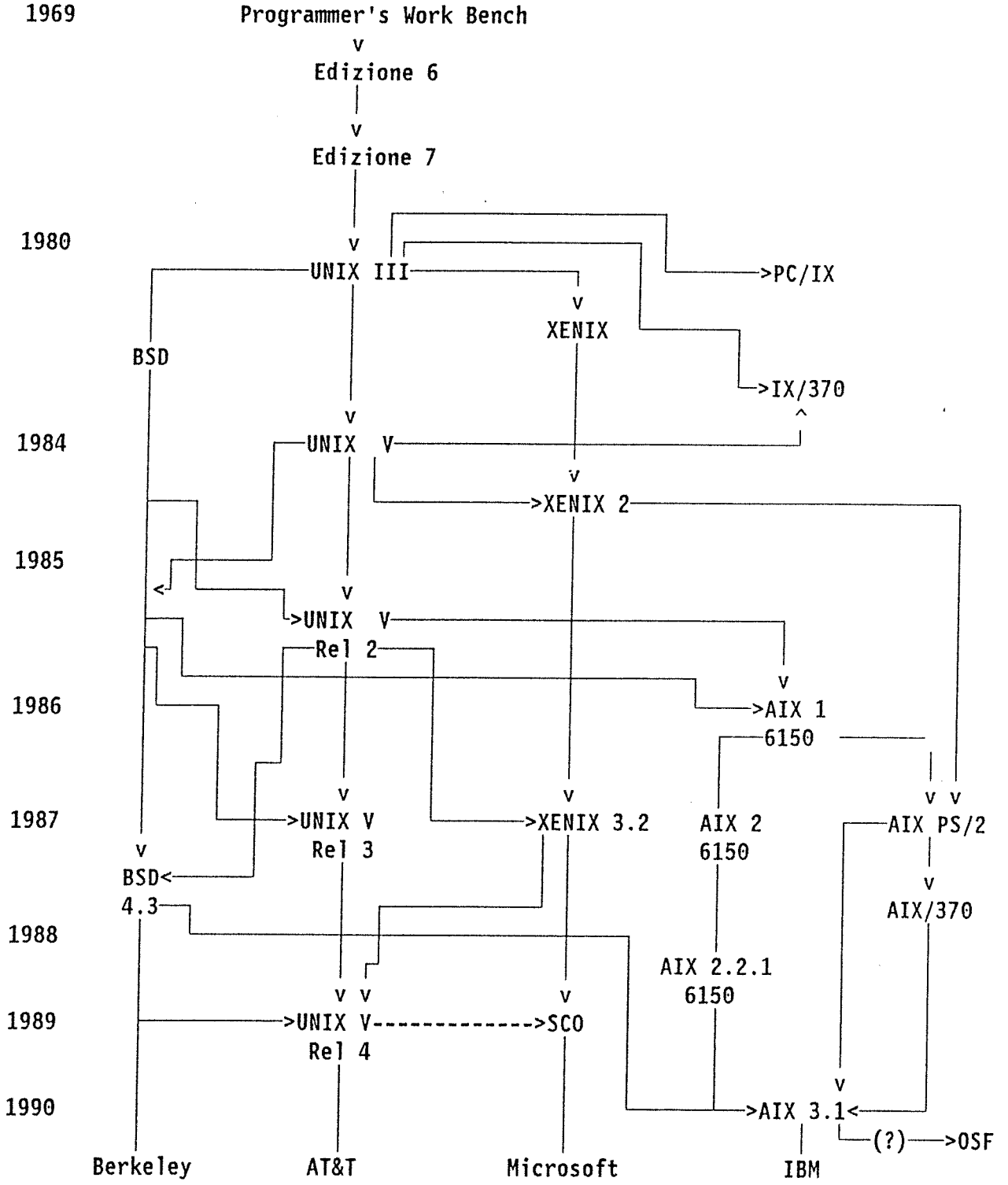
- Nel Gennaio 1983 la AT&T inizia il suo supporto uffciale per UNIX System V, risultato delle aggiunte di nuove funzioni al System III.

- L'Universita' di California, a Berkeley, sviluppa una differente versione di UNIX denominata BSD (Berkeley Software Distribution).

- All'inizio del 1984 esistono circa 100.000 installazioni UNIX nel mondo che girano su macchine che vanno dal mainframe al microprocessore.

- Con la nascita dei microprocessori Motorola 68000, Zilog Z8000, National 32000 e Intel 80x86 il mercato inizia a diffondersi anche in ambienti come uffici, piccoli studi commerciali, applicazioni domestiche o hobbistiche.

L'ALBERO GENEALOGICO di UNIX



- UNIX e' nato nei Bell Laboratories della AT&T.

- Importanti funzioni (cshell, sockets, ...) sono stati sviluppati a Berkeley.

- AIX (Advanced Interactive eXecutive) della **IBM** discende da entrambe le linee di sviluppo ed ha ereditato funzioni sia dal System V che dal BSD.

Introduzione ad AIX

- L'obiettivo principale e' di impadronirsi dei comandi e dei meccanismi fondamentali che permettono di usare il Sistema Operativo AIX.

- Vedremo con quali meccanismi si ottiene l'accesso al sistema, come si fa login e logout, che cosa sono i comandi, che cosa e' una shell.

- Cercheremo di capire cosa sono i meccanismi di piping e di ridirezione e come si possono usare proficuamente.

- Al termine dovremmo essere in grado di accedere il sistema, di usare almeno i comandi piu' semplici, di comprendere la struttura fondamentale di un File System UNIX.

Una sessione con AIX

- Per usare un sistema UNIX occorre che l'amministratore del sistema definisca un identificatore di utente (userid) ed una password.

- Premendo il tasto RETURN sul vostro terminale o sulla console del sistema dovrebbe apparire un messaggio di login

- Fornendo al sistema userid e password si ottiene l'accesso.

Vediamo come si svolge una semplice sessione con AIX.

```
AIX telnet (dedalo)

login: test                                          inserite userid
Password:                                            e password
[YOU HAVE NEW MAIL]                                  c'e' della posta da leggere
$                                                    il sistema e' ora pronto
$                                                    ad accettare i comandi
$ date                                               Ottiene data e ora dal sistema
Wed Mar 27 17:35:47 CET 1991
$ who                                                Chi sta usando la macchina?
mazza       pts/0       Mar 27 10:02
frank       pts/2       Mar 27 10:16
peppe       pts/3       Mar 27 10:18
ferretti    pts/6       Mar 26 17:01
mazza       pts/1       Mar 27 10:06
test        pts/7       Mar 27 17:35          (timeserver.voce.)
montero     pts/8       Mar 27 14:21
test        pts/7       Mar 27 17:33          (timeserver.voce.)
$ mail                                               Leggete la vostra posta
Mail [5.2 UCB] [IBM AIX 3.1]  Type ? for help.
"/usr/spool/mail/test": 1 message 1 unread
>U  1 test     Wed Mar 27 17:33  13/298 "Test"
&
Message  1:
From test Wed Mar 27 17:33 CET 1991
Date: Wed, 27 Mar 91 17:33:59 +0100
From: test
To: test
Subject: Test

Questo e' una nota di test.

& d                                                  cancellate il messaggio
& q                                                  ed uscite
$ mail test                                          Spedite una nota
Subject: Ancora un test
Test
Cc:
$
ctl-d                                                Uscite dal sistema
```

- L'esempio precedente mostra una sessione tipica con AIX 3.1. Al messaggio di login (login:) e' necessario immettere userid e password in lettere **minuscole** .

- Se e' richiesta la password il sistema la chiedera' mediante "Password:" ed essa non apparira' a video.

- Il punto culminante dell'operazione di **login** e' la comparsa del **prompt**, usualmente un singolo carattere, il dollaro ($) o il percentuale (%).

- Il prompt viene inviato da un programma chiamato **Interprete dei comandi** o piu' semplicemente **shell**.

- Puo' essere presente un *Message Of The Day* appena prima del prompt oppure la segnalazione che e' presente della posta.

- Puo' anche venire richiesto che tipo di terminale state usando.

- Da notare il carattere di controllo **ctl-d**: informa il sistema della intenzione di chiudere la sessione.

- Al prompt della shell possono essere inseriti i comandi: sono richieste al sistema.

- Per esempio per ottenere data ed ora dal sistema:

```
$ date
Wed Mar 27 19:00:36 CET 1991
$
```

- Il prossimo comando indica gli utenti che sono logged in:

```
$ who
frank       pts/2        Mar 27 10:16
ferretti    pts/6        Mar 26 17:01
test        pts/1        Mar 27 19:00
$
```

- Ancora "who" sulla vostra userid:

```
$ who am I
test         pts/1        Mar 27 19:00
$
```

- Questo e' il risultato di un errore nell'immissione di un comando:

```
$ whop
whop: not found
$
```

- Errori di battittura nell'immissione dei comandi possono essere corretti per mezzo di caratteri di controllo che spesso dipendono dal tipo di terminale usato. I piu comuni sono l'*erase* character (spesso su **backspace**) e il *line kill* character (spesso su

ctl-u). Possono venir cambiati usando il comando **stty**.

- Il kernel del sistema legge i caratteri man mano che li inserite sul terminale, cosicche' potete battere i comandi alla velocita' che vi e' piu congeniale anche quando non vedete a video il risultato di cio' che battete sulla tastiera (**Type-ahead**).

- L'esecuzione della maggior parte dei comandi puo' essere terminata premendo il tasto di interruzione (anche esso modificabile con stty). L'assegnazione piu' comune e' sui tasti di **ctl-c**, **ctl-BREAK**. Un altro modo (molto drastico) di terminare l'esecuzione dei programmi e quello di spegnere il terminale su cui state lavorando (o comunque di interrompere la comunicazione con il sistema operativo).

- Con il carattere **ctl-s** si puo' sospendere l'output a video di un programma e il carattere **ctl-q** provvedera' a ripristinare la normale esecuzione.

- Il modo piu' semplice di concludere una sessione e' quello di premere il carattere **ctl-d**. Altrettanto bene

(e in modo piu' sicuro) si puo immettere il comando **exit**.

Comandi e Manuale

- Tutti i comandi sono documentati nel **Commands Reference**.

- Spesso il manuale e' mantenuto on-line in modo che ogni utente lo possa consultare a video (e' presente per AIX PS/2, AIX/370 e AIX 3.1). Per esempio per avere informazioni sull'uso di **who** e' sufficiente dare il comando "**man who**" e ovviamente una delle prime cose da provare e' "**man man**".

- Questo e' un esempio di pagina di manuale on-line, riferita al comando **head** (da AIX PS/2 1.2):

head

PURPOSE

Prints the first few lines of a file or files.

SYNTAX

```
        +---------+
head ---|         |--- file ---|
        +- count -+
```

DESCRIPTION

The head command prints the first count lines of each of the specified files, or of the standard output. If count is omitted, it defaults to 10.

RELATED INFORMATION

See the following command: "tail."

Introduzione al File System

- Le informazioni nel sistema AIX sono memorizzate in *files*. Ad ogni file viene assegnato un nome, dello spazio per mantenere i dati, e delle informazioni che ne permettono la amministrazione.

- Un file puo' contenere qualunque tipo di informazione e dal punto di vista del sistema non e' altro che una stringa di bytes.

- Il File System e' organizzato in modo che ogni utente possa avere i suoi file personali senza dover interferire con i file di proprieta' di altre persone. Molti dei comandi di AIX sono dedicati alla gestione ed amministrazione dei files.

Creazione di files

- La prima cosa necessaria (e forse la piu' importante in ogni sistema operativo) e' imparare a creare e modificare i files.

- L'editor di base in ambiente UNIX e' **ed**. Cio' significa che in ogni installazione e' disponibile ed: e' un editor di linea, non necessita' di un terminale particolare ed e' la base di altri comandi.

- Per esempio per creare un file:

```
$ ed prova                                  Esegue l'editor ed
?prova                                      il file non esiste!
a                                           comando ed per aggiungere
provate a scrivere qualche cosa
qualunque cosa desideriate scrivere ...
.                                           termina il modo append
w prova                                     scrive nel file prova
73                                          numero di caratteri
q                                           quit da ed
$
```

- Il comando a ("**append**") indica all'editor di iniziare a salvare il testo che state scrivendo. Il "." segnala la fine del testo e deve essere inserito all'inizio della linea.

- Il comando w ("**write**") memorizza le informazioni che avete inserito nel file chiamato *prova*. Il *filename* puo' essere qualunque parola di vostro gradimento.

- ed risponde indicando il numero di caratteri salvati nel file. Fino a che non date il comando "w" niente viene memorizzato, risiede solo nell'area dati del programma ed.

- Se a questo punto voleste modificare il file potete accederlo dando ancora il comando **ed prova**.

- Proviamo ora a creare un paio di file ed ad ottenere l'elenco dei nostri files personali:

```
$ ed
a
Questo e' il file test ...
.
w test
27
q
$
$
$ ed
a
... e questo e' il file prova
.
w prova
30
q
$
```

- Il comando **ls** (list) elenca i nomi (non il contenuto dei files):

```
$ ls
prova  test
$
```

- *ls*, cosi' come la maggior parte dei comandi, ha delle **opzioni** che possono essere usate per

modificare il suo comportamento di default. Le opzioni seguono il nome del comando sulla command line e sono usualmente preceduti dal segno meno "-". Per esempio **ls -l** produce un elenco piu' dettagliato delle informazione relative a ciascun file:

```
$ ls -l
total 8
-rw-r--r--   1 test     staff          30 Mar 28 16:11 prova
-rw-r--r--   1 test     staff          27 Mar 28 16:10 test
$
```

- "total 8" indica quanti blocchi di spazio disco occupano i nostri files (blocchi usualmente da 512 o 1024 bytes). La stringa -rw-r--r-- indica chi ha il permesso di leggere e scrivere il file. In questo caso il proprietario ("test") puo' leggere e scrivere, gli altri possono solo leggere. Il numero "1" che segue e' il numero di "link" del file. 30 e 27 sono il numero dei caratteri presenti nei files corrispondenti, che sono in accordo con i numeri ottenuti da ed. La data e l'ora indicano l'ultima volta che il file e' stato **modificato**.

- Le opzioni possono venire raggruppate: **ls -lt** da' lo stesso risultato di **ls -l**, ma ordinato per data. L'opzione **-u** produce informazioni sull'accesso ai file: **ls -lut** da un long list ordinato per l'utilizzo piu' recente. L'opzione **-r** inverte l'ordine dell'output cosi', **ls -rt** lista i file per ordine data, ma dal piu' vecchio al piu' recente.

- Ovviamente potete indicare ad ls il nome del file a cui siete interessati. Cosi':

```
$ ls -l test
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
$
```

- La stringhe che seguono il nome del programma sulla command line, cosi' come *-l* e *test*, vengono chiamate **argomenti** del programma. Gli argomenti sono usualmente opzioni o nomi di files che devono essere usati dal programma stesso. In generale, se un comando accetta delle opzioni esse precedono qualsiasi filename, ma possono, generalmente, apparire in qualunque ordine.

- Queste sono regole di uso generale, ma occorre porre attenzione al fatto che ogni comando possiede le sue idiosincrasie e cio' puo', a volte, sconcertare chi si accinge ad usare il sistema per la prima volta.

Visualizzazione di files

- Ora che abbiamo creato dei files nasce il problema di visualizzarne il contenuto a video. Esistono molti programmi che svolgono questo compito. Una possibilita' e' usare l'editor:

```
$ed test                          ed riporta 27 caratteri in test
27
1,$p                              Stampa dalla riga 1 all'ultima
Questo e' il file test ...        Il file ha solo una riga
q                                 Ed esce dall'editore
$
```

- ed inizia riportando il numero di caratteri presenti nel file *test*. Il comando **1,$p** indica di stampare tutte le righe presenti nel file.

- A volte non e' comodo usare un editor (ed e' anche uno spreco) semplicemente per ottenere la stampa di un file sul display del terminale. Inoltre cio' ci

limiterebbe ad un solo file per volta, mentre puo' essere necessario stamparne diversi.

- Una alternativa e' il comando **cat** (uno dei piu' usati comandi sotto shell: e' in qualche modo paragonabile al comando **type** del PCDOS). **cat** stampa a video il contenuto di tutti files elencati come argomenti:

```
$
$ cat test
Questo e' il file test ...
$ cat prova
... e questo e' il file prova
$ cat test prova
Questo e' il file test ...
... e questo e' il file prova
$
```

- I files elencati come argomento vengono *concatenati* (in inglese *Catenate* una abbreviazione di "concatenate"), sul terminale, senza che venga inserito nulla tra l'uno e l'altro (vedremo che cio' e' molto importante nella struttura dei comandi)

- Non c'e' nessun problema con files brevi, ma ctl-s puo' aiutare a visualizzare files piu' lunghi su un terminale "veloce".

- Un altro comando utile per visualizzare files o stamparli su una printer e' **pr**. Cosi' come "cat", "pr" stampa il contenuto dei file, ma in una forma piu' adatta per una stampante: ogni pagina e' lunga 66 righe (11 pollici, il formato A4), con la data e l'ora dell'ultimo cambiamento apportato al file, il numero di pagina e il filename su ogni pagina.

```
$ pr test prova


Thu Mar 28 16:10:41 1991  test Page 1


Questo e' il file test ...
     (60 o piu' righe bianche)


Thu Mar 28 18:14:12 1991  prova Page 1


... e questo e' il file prova
     (60 o piu' righe bianche)

$
```

- **pr** puo' stampare files anche su piu' colonne. Per esempio **pr -3** *filenames* stampa ogni file su 3 colonne.

Copiare, Rinominare e Cancellare files

- Ora abbiamo dei files disponibili, sappiamo crearli, visualizzarli, modificarli e stamparli. Nasce subito il problema di ottenere delle copie, di rinominarli e, se non sono piu' necessari, di cancellarli.

- Vediamo subito come si copia un file. Ricordate il file "test" creato nella sezione precedente. Cerchiamo di ottenerne una copia:

```
$
$ ls -l test
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
$ cp test copia_di_test
$
$ ls -l
total 16
-rw-r--r--   1 test     staff         27 Mar 29 10:46 copia_di_test
-rw-r--r--   1 test     staff        306 Mar 28 19:53 ed
-rw-r--r--   1 test     staff         30 Mar 28 18:14 prova
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
$
```

- Il comando "**cp test copia_di_test**" (copy) copia il file "test" nel file "copia_di_test". Il comando "ls" ci ha permesso di verificare che l'operazione e' andata a buon fine.

- Ora rinominiamo il file "copia_di_test". In UNIX esiste il concetto di **muovere** un file all'interno di un file system:

```
$
$
$ mv copia_di_test nuovo
$
$ cat copia_di_test
cat: cannot open copia_di_test
$
$ cat nuovo
Questo e' il file test ...
$
```

- L'esempio precedente ci mostra che ora il file "copia_di_test" non esiste piu', mentre al suo posto esiste il file "nuovo" che e' una sua copia.
  Attenzione, se *muovete* con il comando **mv** (move) su un nome che esiste gia' il file target verra' rimpiazzato (e quindi ne perderete il contenuto).

- Cancelliamo ora la copia del file "test" il cui ultimo nome e' "nuovo". Il comando usato per cancellare files e' **rm** (remove):

```
$
$ ls -l
total 16
-rw-r--r--   1 test     staff        218 Mar 29 10:57 ed
-rw-r--r--   1 test     staff         27 Mar 29 10:46 nuovo
-rw-r--r--   1 test     staff         30 Mar 28 18:14 prova
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
$
$ rm nuovo
$
$ ls -l
total 12
-rw-r--r--   1 test     staff        218 Mar 29 10:57 ed
-rw-r--r--   1 test     staff         30 Mar 28 18:14 prova
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
$
```

- Quindi abbiamo cancellato il file "nuovo" con il comando "**rm nuovo**" ed il comando "ls" ci mostra chiaramente il risultato dei comandi che abbiamo dato fino ad ora.

- Il comando "rm" puo' cancellare piu' files dati come argomenti e possiede un opzione che rende la cancellazione piu' sicura. In particolare l'opzione

"**-i**" (interactive) richiede la conferma per ogni file di cui e' stata richiesta la cancellazione:

```
$ cp test test1
$ cp test test2
$
$ rm -i test1 test2     (opzione -i)
test1: y                (conferma la cancellazione)
test2: n                (la annulla)
$ ls -l
total 16
-rw-r--r--   1 test     staff         577 Mar 29 11:02 ed
-rw-r--r--   1 test     staff          30 Mar 28 18:14 prova
-rw-r--r--   1 test     staff          27 Mar 28 16:10 test
-rw-r--r--   1 test     staff          27 Mar 29 11:06 test2
$
```

- Abbiamo imparato a **creare**, **visualizzare**, **copiare**, **cancellare** e **rinominare** files. Siamo ora piu' padroni del sistema e siamo in grado di eseguire alcune operazioni fondamentali. Chi programma usa, per la maggior parte del tempo, proprio questi comandi per svolgere il suo lavoro quotidiano.

- E' arrivato pero' il momento di comprendere meglio come e' strutturato il File System.

Files e Directories

- Abbiamo gia' usato i **Files** di AIX per alcuni semplici esperimenti. Ma cosa e' un FILE? Dal punto di vista del sistema un file e':

> **"Una sequenza di bytes"**

- I files di UNIX non hanno una struttura predefinita (quindi senza il concetto di record). E' quindi una delle strutture piu' semplici che si possano pensare.

- Il contenuto di un file ha un significato solo per i programmi che lo usano.

- In genere sono memorizzati su disco magnetico.

- Ad ogni file e' associato un **filename** (il nome del file). E' una stringa di caratteri che puo' essere scelta in quasi assoluta liberta'. Il senso comune suggerisce pero' di usare solo caratteri visualizzabili a terminale e che non coincidano con i metacaratteri usati dalla shell o da altri programmi di utilita' ("*", ".", "-" etc.).

- La lunghezza del filename puo' raggiungere i **255** caratteri su **RISC/6000** e **PS/2**, ma per compatibilita' con il sistema UNIX e consigliabile non superare i **14** caratteri (per esempio sul **6150** sotto **AIX RT 2.2.1**).

- Ricordate sempre che AIX distingue le lettere minuscole dalle maiuscole (non e' come il PCDOS). Il file "test" e quindi differente dal file "TEST".

```
$
$ cat >TEST
Questa e' una riga del file TEST.
$ (ctl-d)
$
$ ls -l
total 20
-rw-r--r--   1 test     staff         34 Mar 29 12:15 TEST
-rw-r--r--   1 test     staff        510 Mar 29 11:07 ed
-rw-r--r--   1 test     staff         30 Mar 28 18:14 prova
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
-rw-r--r--   1 test     staff         27 Mar 29 11:06 test2
$ cat test
Questo e' il file test ...
$ cat TEST
Questa e' una riga del file TEST.
$
```

- E' probabile che in un ambiente 'multi-user' si verifichino omonimie, ovvero che piu' utenti si trovino ad usare lo stesso filename per files differenti.

- UNIX risolve questo problema raggruppando i files in **directories**. Una "directory" e' un file (si' e' proprio cosi': anche essa e' un file, diciamo un po' particolare) che contiene informazioni relative agli altri files (un po' come un indice).

- Una directory puo' contenere files ordinari oppure altre directories. Cio' permette di formare una struttura **gerarchica**. Il modo piu' naturale per immaginarla e come un **albero** di directories e files. E' possibile muoversi attraverso questo albero e individuare ogni file nel sistema partendo dalla **radice** (*root* in inglese) e muovendosi lungo i diversi rami. Non esiste nessuna limitazione di livello.

- Ogni utente ha una directory *personale*, la **home directory** (chiamata anche **login directory**), che contiene solo files che gli appartengono. Quando fate login vi trovate nella vostra "home directory" . Potete cambiare la directory in cui lavorate (chiamata **current directory**), ma la vostra *home directory* rimane sempre la stessa.

- Sperimentiamo questo concetto. Il comando da cui partiamo e' **pwd** ("print working directory"). che visualizza il nome della della directory in cui vi trovate (la *current directory* appunto):

```
$
$ pwd
/u/test
$
```

Il risultato del comando vi informa che siete nella directory **test** che e' nella directory **u** che e' posizionata a sua volta nella **root directory**. La radice viene convezionalmente denominata **/**. Il carattere '/' (barra o slash) separa anche i componenti del nome (**pathname**). Il limite dei 255 (o 14) caratteri si applica solo ad ogni singolo componente di tale nome.

- Tutti i comandi che abbiamo esaminato fino ad ora possono agire su un "filename" o su un "pathname". Per esempio il comando **"ls -l /u/test"** otterra' un *long list* della directory "/u/test" (la home directory dell'utente "test" con cui abbiamo acceduto il sistema).

```
$
$ ls -l /u/test
total 12
-rw-r--r--   1 test     staff         90 Mar 29 12:31 ed
-rw-r--r--   1 test     staff         30 Mar 28 18:14 prova
-rw-r--r--   1 test     staff         27 Mar 28 16:10 test
$
```

- Proviamo ora ad ottenere un list della directory "/u" (che contiene le "home directories" degli utenti del sistema:

```
$
$ ls /u
aix12          cmpsl          langmod        montero        tangx
alto           federico       local          pacelli        tarricon
bin            ferretti       lost+found     peppe          test
bin.PCRT       ferri          man.PS2        scarci         verola
bin.PS2        frank          massimo        src            xbook
bin.RS6000     guest          mazza          tangora        xst
cbase          itest          minosse        tangora.save   zaza
$
```

- Ed ora un listing della directory *root* (da un AIX 3.1 su un RISC/6000):

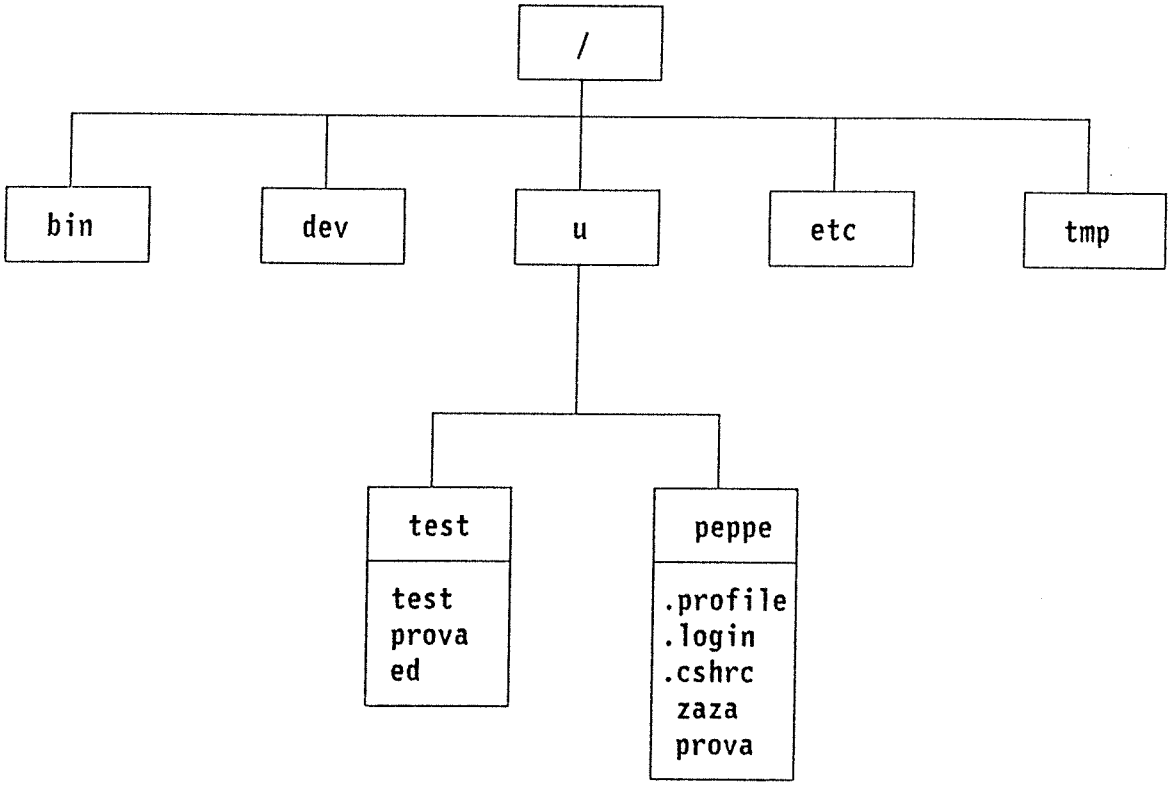
```
$ ls -l /
total 2712
-rwxr-xr-x   1 root     system        99 Feb 12 14:31 backu
drwxr-xr-x   2 bin      bin         3584 Nov 15 20:06 bin
drwxrwxrwx   2 root     system       512 Aug 01 1990  blv
-r-xr-xr-x   1 root     system        56 May 19 1990  bootrec
drwxrwxr-x   2 root     system      3072 Mar 25 13:55 dev
drwxrwxr-x  20 root     system      6144 Mar 29 09:47 etc
lrwxrwxrwx   1 root     system         2 Feb 12 20:00 home -> /u
drwxr-xr-x   3 bin      bin          512 Aug 01 1990  info
drwxr-xr-x   5 bin      bin         1024 Apr 23 1990  lib
drwx------   2 root     system       512 Aug 01 1990  lost+found
-rw-------   1 root     system      2124 Sep 07 1990  mbox
drwxr-xr-x   2 bin      bin          512 Aug 01 1990  mnt
drwxr-xr-x   2 root     system       512 Feb 27 16:00 rtmarco
-rw-r--r--   1 root     system    519788 Mar 26 10:33 smit.log
-rw-r--r--   1 root     system     22570 Mar 26 10:33 smit.script
drwxr-xr-x   2 root     system       512 Aug 03 1990  supplies
drwxr-x---   6 tangora  voce         512 Mar 13 11:46 tang
drwxr-x---   4 tangora  voce         512 Mar 20 11:08 tangora
drwxrwxrwt   5 bin      bin         1024 Mar 29 14:00 tmp
drwxr-xr-x  36 root     system      1024 Mar 26 10:33 u
-r-xr-xr-x   1 root     system   1261933 May 30 1990  unix
-r-xr-xr-x   1 root     system    887159 Aug 01 1990  unix.strip
drwxr-xr-x  26 bin      bin          512 Jan 30 10:51 usr
drwxr-xr-x   2 root     system       512 Aug 03 1990  waste
-rw-r--r--   1 bin      bin            0 Feb 25 19:11 xtalk.names
$
```

- Il **pathname** rappresenta il nome completo di ogni file nel sistema (e lo distingue univocamente). **cat** come "ls" puo' agire tanto su pathnames che su filenames (cosi' come qualunque altro comando: questa e' una caratteristica di una funzione di base del sistema operativo, la *open()*). Cosi':

```
$
$ cat /u/test/prova
... e questo e' il file prova
$
```

visualizza il file "prova" che si trova nella "home directory" dell'utente "test".

- Il file system e' organizzato come un albero genealogico. Questa e' una immagine di una parte del file system dell'RS/6000 che abbiamo listato prima:



- I pathname non sono molto interessanti se i files si trovano nella directory di vostra proprieta', ma se intendete lavorare con i file di altre persone o altri progetti essi divengono necessari. Per esempio i vostri colleghi possono accedere al file "test" con il comando **cat /u/test/prova**. Analogamente potreste accedere il file "prova" nella directory "/u/peppe" con il comando **cat /u/peppe/prova**.

- Da notare i due significati del caratttere "*/*": indica la **root** directory e separa i filenames.

Cambiare Directory

- Il prossimo passo e' quello di cambiare directory: il comando fondamentale e' **cd** (*changing directory*). Il comando cd accetta come parametro il pathname della directory in cui volete spostarvi, ecco un esempio:

```
$ pwd
/u/test
$
$ cd /u/guest
$
$ pwd
/u/guest
$
$ cd
$
$ pwd
/u/test
$
```

- Nell'esempio partendo dalla **home directory** dell'utente test (/u/test) il comando **cd** ci ha permesso di spostarci nella directory **/u/guest**. Il comando cd senza argomento ci ha fatto ritornare nella home directory.

- Ancora un altro esempio:

```
$
$ pwd
/u/test
$ ls
prova  test
$ mkdir esempio            crea la directory esempio
$ ls
esempio  prova    test
$ cd esempio               si sposta in esempio
$ pwd
/u/test/esempio
$ cd ..                    ritorna indietro di un livello
$ pwd
/u/test
$ rmdir esempio            rimuove la directory esempio
$ ls
prova  test
$
```

- I caratteri "**..**" indicano la directory *padre* di qualsiasi directory, ovvero la directory che e' di un livello piu' vicina alla radice del file system. Invece "**.**" indica **sempre** la **current directory**. Ancora una volta abbiamo usato **cd** senza argomenti per ritornare nella **home directory**.

- Abbiamo introdotto due nuovi comandi. **mkdir** crea una nuova directory e **rmdir** la rimuove (purche' non contenga dei files).

Altri Comandi utili

- Aggiungiamo qualche altro comando utile ai nostri strumenti. Il comando **wc** (*word count*) conta il numero di righe, di parole e di caratteri in uno o piu' files. Prendiamo come file di lavoro il file *storia.unix* nella current directory dell'utente *test*

```
$
$ cat storia.unix
Il gruppo del progetto MULTICS viene lasciato
senza un ambiente di lavoro.

Thompson e Ritchie disegnano un nuovo File System
che si evolvera' in una delle prime versioni del
File System di UNIX.

Thompson scrive un programma che simula il File
System proposto e codifica un Kernel preliminare
per un GE 645.
$
```

- Ora applichiamo il comando word count a questo file:

```
$
$ wc storia.unix
      10      52     308 storia.unix
$
```

Il comando ci informa che il file storia.unix contiene 10 linee, 52 parole e 308 caratteri. La definizione di "parola" e' molto semplice: qualsiasi stringa che non contenga un blank, un tab (0x09) un ritorno a capo (line feed 0x0a). wc puo contare su piu' files e fornire anche i totali:

```
$
$ wc prova test
       1       7      30 prova
       1       6      27 test
       2      13      57 total
$
```

- Ora il comando *grep*: ricerca nei files le linee che contengono un pattern (il nome proviene dal comando g/*regular-expression*/p di **ed**). Cerchiamo la stringa "Thomposon" in "storia.unix":

```
$
$ grep Thompson storia.unix
Thompson e Ritchie disegnano un nuovo File System
Thompson scrive un programma che simula il File
$
```

Ricerchiamo le linee in cui non compare la stringa "Thompson". Il flag **-v** rovescia il senso della ricerca.

```
$
$ grep -v Thompson storia.unix
Il gruppo del progetto MULTICS viene lasciato
senza un ambiente di lavoro.

che si evolvera' in una delle prime versioni del
File System di UNIX.

System proposto e codifica un Kernel preliminare
per un GE 645.
$
```

- Riepiloghiamo in una tabella i comandi usati fino ad ora:

Table 1 (Page 1 of 2). Alcuni dei comandi piu' usati

Comando	**Funzione**
ls	Elenca i nomi dei files nella current directory
ls *filenames*	Elenca i files dati come argomenti
ls -t	Elenca in ordine di data; prima il piu' recente
ls -l	Elenco dettagliato
ls -u	Elenca prima il file acceduto piu' recentemente
ls -r	Elenco in ordine rovesciato: -rt, -rlt, etc
ed *filename*	edit del file in argomento
cp *file1 file2*	copia *file1* su *file2*, sostituisce *file2* se esiste gia'
mv *file1 file2*	muove (rinomina) *file1* su *file2*, sostituisce *file2* se esiste gia'
rm *filenames*	rimuove (cancella) i files dati come argomento
cat *filenames*	visualizza il contenuto dei files dati come argomenti
pr *filenames*	stampa il contenuto con l'intestazione, 66 righe per pagina
pr -n *filenames*	stampa su n colonne

pr -m *filenames*	stampa i files uno accanto all'altro su piu' colonne
wc *filenames*	conta le linee, le parole e i caratteri per ognuno dei files
wc -l *filenames*	conta le linee per ognuno dei files
grep *pattern filenames*	stampa ogni linea in cui compare *pattern*
grep -v *pattern filenames*	stampa ogni linea in cui **non** compare *pattern*
sort *filenames*	pone in ordine alfabetico le linee dei files in argomento
tail *filename*	stampa le ultime 10 linee del file
tail -n *filename*	stampa le ultime n linee del file
tail +n *filename*	stampa il file a partire dalla linea n
cmp *file1 file2*	stampa la posizione della prima differenza tra i files
diff *file1 file2*	stampa *tutte* le differenze tra i files

La Shell

- Abbiamo visto che il punto culminante dell'operazione di login e la comparsa del **prompt**, nel nostro caso la comparsa a terminale del carattere "**$**". Da questo momento possono venire immessi i comandi.

- Non state comunicando direttamente con il kernel del sistema: i vostri comandi vengono interpretati da un programma, un *interprete dei comandi* (qualcosa di simile al COMMAND.COM o al CMD.EXE del PCDOS e di OS/2 per chi li conosce gia').

- Il programma che esegue questo compito viene denominato **shell**. La *shell* e' un normalissimo programma simile a *date* o *who*, sebbene sia molto importante. Il suo scopo e' quello di tradurre le stringhe che battete sulla tastiera in operazioni che il sistema operativo sia in grado di eseguire.

- I suoi principali obiettivi sono:

  - Abbreviazione per i *filenames*: operare su un insieme di files specificando un pattern per i nomi - la shell ricerchera' i files nei quali compare il pattern specificato.

  - Ridirezione dell'input-output: dirigere l'output di un programma su un file invece che sul terminale e specificare che l'input proviene da un file invece che dalla tastiera. L'output di un programma puo' venire connesso anche con l'input di un altro programma (*piping*).

  - Personalizzazione dell'ambiente: definire i propri comandi e le proprie abbreviazioni.

Espansione dei Filenames

- Accade spesso di dover lavorare con files i cui filenames possono venir raggruppati in qualche modo. Per esempio i sorgenti di un vostro programma C termineranno tutti in *.c* oppure i capitoli del libro che state scrivendo inizieranno tutti per *cap*.

- Supponiamo che il programma C a cui state lavorando si trovi nella subdirectory **source** della vostra home directory e che i files che lo compongono siano *progr.c subr1.c subr2.c subr3.c incl1.h incl2.h*. Nella subdirectory *doc* si trova invece la documentazione relativa al programma suddivisa nei files *doc1.1 doc1.2 doc1.3 doc2.1 doc2.2 doc2.3 doc2.4* dove i numeri si riferiscono a capitoli e sezioni.

- Un modo per stampare la documentazione relativa al secondo capitolo potrebbe essere:

```
$
$ pr doc2.1 doc2.2 doc2.3 doc2.4
$
```

E' abbastanza chiaro che se il numero dei capitoli e' appena piu' grande di 4 la cosa puo' diventare macchinosa e diventa abbastanza facile commettere errori.

- Per fortuna la **shell** svolge gran parte del lavoro al nostro posto. L'alternativa e' infatti quella di usare i carattere **wild-card** o **metacaratteri**. Infatti il seguente comando esegue in maniera piu' corretta il lavoro:

```
$
$ pr doc2.*
$
```

E se volessi stampare tutta la documentazione:

```
$
$ pr doc*
$
```

- I metacaratteri sostituiscono i filenames, o parti di questi, e consentono di creare una corrispondenza tra l'abbreviazione e i nomi estesi. Il carattere "*" puo' sostituire qualunque stringa compresa la stringa nulla. Il carattere "**?**" sostituisce un qualsiasi singolo carattere. Le parentesi quadre "**[ ]**" permettono di rappresentare un insieme di caratteri. Per esempio **[a-z]** e' la notazione abbreviata per le lettere minuscole dell'alfabeto.

- Il punto cruciale e' che l'abbreviazione del filename non e' una proprieta' del comando **pr**, ma un funzione svolta dalla **shell**. La shell interpreta il carattere "*" come *qualsiasi stringa di caratteri* cosicche' "doc*" e' un pattern che corrisponde a tutti i filenames nella current directory che iniziano con la stringa "doc".

- La shell (non il sistema operativo) crea la lista, in ordine alfabetico, dei filenames che fanno match con il pattern e la passa come argomento al comando pr.

- Per chi proviene dal mondo PCDOS (o OS/2) questa e' una delle differenze piu' importanti. Il COMMAND.COM del PCDOS non svolge questa funzione per tutti i programmi, ma solo per alcuni comandi interni (come dir).

- Si posssono usare i **wildcard** con qualsiasi comando per generare un elenco di filenames. Per esempio per contare il numero di parole nel secondo capitolo del manuale del vostro programma:

```
$
$ wc doc2.*
    360    2017   15100 doc2.1
    132     297    3536 doc2.2
    139     763    6780 doc2.3
    360    2017   15100 doc2.4
    991    5094   40516 total
$
```

- Il programma **echo** e' particolarmente importante quando si vuole sperimentare con l'espansione dei metacaratteri. Come potete supporre l'unico compito svolto da **echo** e' di fare l'eco di tutti i suoi argomenti. Cosi':

```
$
$ echo Lista di Argomenti
Lista di Argomenti
$
```

- In questo caso **echo** ha prodotto semplicemente l'eco degli argomenti che gli abbiamo forniti. Ma gli argomenti possono venire generati dall'espansione dei caratteri wild-card:

```
$
$ echo doc2.*
doc2.1 doc2.2 doc2.3 doc2.4
$
```

La shell ha espanso l'abbreviazione "**doc2.***" nei filenames che corrispondono al secondo capitolo del manuale producendo l'elenco **doc2.1 doc2.2 doc2.3**

doc2.4 per il comando **echo** esattamente come ha fatto per il comando **wc** dell'esempio precedente.

- Se volessimo elencare tutti i files della directory corrente in ordine alfabetico non dovremmo fare altro che dare il comando:

```
$
$ echo *
doc1.1 doc1.2 doc1.3 doc2.1 doc2.2 doc2.3 doc2.4
$
```

cosi' come il comando "**pr** *" stampera' tutti i files della documentazione (in ordine alfabetico).

- **Attenzione** a comandi come **rm** * : rimuovera' tutti i files della vostra directory corrente.

- Qualche esperimento con i metacaratteri ed echo. L'uso del "**?**"

```
$
$ echo doc?.2
doc1.2 doc2.2
$
```

ed ora l'uso delle parantesi quadre **[ ]** per rappresentare un range di caratteri.

```
$
$ echo doc2.[1234]
doc2.1 doc2.2 doc2.3 doc2.4
$ echo doc2.[1-4]
doc2.1 doc2.2 doc2.3 doc2.4
$ echo doc?.[1-4]
doc1.1 doc1.2 doc1.3 doc2.1 doc2.2 doc2.3 doc2.4
$ echo doc[12].[1-4]
doc1.1 doc1.2 doc1.3 doc2.1 doc2.2 doc2.3 doc2.4
$
```

- Da notare che i metacaratteri trovano corrispondenza solo in files che esistono. In particolare non si possono creare nuovi files usando i metacaratteri. Per esempio "**mv doc1.* cap1.***" non vi permettera' di rinominare i files sia perche' cap1.* non si riferisce a files che esistno gia' ed anche perche' il comando mv non funziona in questo modo (attenzioni utenti PCDOS, mv non e' equivalente al rename).

- In alcuni sistemi UNIX esiste un limite massimo alla lunghezza di un comando. Se esistono troppi files che soddisfano il criterio di selezione si puo' incorrere in un errore.

- I metacaratteri possono essere usati nei **pathnames** cosi' come nei **filenames**. Se nella directory /u/test esiste una copia della documentazione nella directory doc.copy si possono elencare i files di doc e doc.copy con un solo comando:

```
$
ls test/doc*/doc1.[12]
test/doc.copy/doc1.1  test/doc/doc1.1
test/doc.copy/doc1.2  test/doc/doc1.2
$
```

- **Puntualizziamo ancora una volta che e' la shell ad interpretare i metacaratteri.** Qualunque comando invocato dalla shell riceve gli argomenti prodotti dall'espansione dei metacaratteri (cio' semplifica notevolmente la programmazione ed e' indicativo dell'eleganza del disegno di UNIX).

- Puo' essere necessario impedire all shell di interpretare i metacaratteri. Cio' puo' essere ottenuto includendo gli argomenti di comandi tra singoli apici ' o tra doppi apici ".

- I due metodi sono differenti: i singoli apici impediscono alla shell di interpretare qualsiasi metacarattere, mentre i doppi apici le permettono di interpretare i caratteri $, `, \ percio' usate i singoli apici ' a meno che non vogliate esplicitamente che la shell interpreti questi caratteri.

```
$
$ echo doc?.1
doc1.1 doc2.1
$
$ echo "doc?.1"
doc?.1
$
```

- Infine il **backslash ** impedisce che il carattere seguente venga interpretato dalla shell:

```
$
$ echo doc?.1
doc1.1 doc2.1
$
$ echo doc\?.1
doc?.1
$
```

Ridirezione dell'Input/Output

- La shell permetti di rimpiazzare il terminale con un **file** sia per l'input che per l'output. Cosi **ls** elenca i files della directory corrente su terminale, mentre **ls > elenco** produce l'elenco nel file *elenco*.

- Il simbolo > indica alla shell di memorizzare lo **Standard Output** di qualsiasi comando nel file che segue. Il file verra' creato se non esiste o sara' riscritto se esisteva gia'. Nulla viene prodotto sul terminale.

- Questo disegno permette di usare cat per, appunto, **concatenare** i files. Cosi per ottenere in unico file il primo capitolo del manuale potreste usare:

```
$
$ wc doc1.*
     139     763    6780 doc1.1
     396    1840   18036 doc1.2
     264    1014   10793 doc1.3
     799    3617   35609 total
$
$ cat doc1.* >cap1
$
$ wc cap1
     799    3617   35609 cap1
$
```

- Il simbolo >> svolge lo stesso compito ma piuttosto di riscrivere il file (se esiste gia') appende alla fine del file l'output del programma in questione.

```
$
$ cat test
Questo e' il file test ...
$ cat prova
... e questo e' il file prova
$ cat output
File di output ...
$ cat test prova >>output
$ cat output
File di output ...
Questo e' il file test ...
... e questo e' il file prova
$
```

- Nello stesso modo il simbolo < indica alla shell di sostituire lo **Standard Input** (la vostra tastiera) con il file che segue.

- Se voleste contare il numero degli utenti che sono login nel vostro sistema potreste usare la seguente sequenza di comandi:

```
$
$ who >temp
$ wc -l <temp
       6
$
```

oppure contare il numero dei files presenti nella current directory:

```
$
$ ls >temp
$ wc -l <temp
      14
$
```

o ancora individuare se un vostro collega e' login sul sistema:

```
$
$ who >temp
$ grep frank <temp
frank       pts/1       Apr 17 17:50
$
```

- I comandi **cat test** e **cat <test** svolgono lo stesso compito, ma esiste una importante differenza concettuale: nel primo caso test viene interpretato dalla shell e passato a cat come argomento, mentre nel secondo cat non vede nessun argomento. Si da' semplicemente il caso che cat sia disegnato per leggere il suo Standard Input quando non gli vengono forniti argomenti.

- Questa e' una opzione che viene fornita dalla maggior parte dei comandi (ed e' da tenere in mente quando se ne scrivono di nuovi): se non sono presenti filenames sotto forma di argomenti viene processato lo Standard Input. E' in questo modo che abbiamo potuto usare cat come un semplice editor in uno degli esempi precedenti (e qualche volta capita realmente di usarlo cosi').

Concatenazione di comandi

- In alcuni degli esempi precedenti e' accaduto di utilizzare l'output di un programma come input per un altro comando. Questa neccessita' puo' manifestarsi frequentemente, tanto spesso da meritare l'introduzione di un nuovo meccanismo nel sistema UNIX: quello delle **pipelines**.

- Una **pipe** e' il mezzo per connettere lo Standard Output di un programma con lo Standard Input di un altro, senza la necessita' di usare un file temporaneo. Una **pipelines** e' la concatenazione di due o piu' programmi attraverso pipes. La barra verticale, il carattere "|".

- Rivediamo alcuni degli esempi precedenti usando le pipes invece di un file temporaneo. Contiamo il numero di utenti:

```
$
$ who | wc -l
       7
$
```

Contiamo il numero di files nella directory corrente:

```
$
$ ls | wc -l
      15
$
```

Ed ora cerchiamo un utente login sul nostro sistema:

```
$
$ who | grep frank
frank        pts/1        Apr 17 17:50
$
```

- Concatenando i comandi per mezzo delle pipes a formare una pipeline si posssono risolvere una grande varieta' di problemi riusando cio' che e' disponibile, senza dover implementare nuovi comandi. Cio' richiede che i comandi base siano

ragionevolmente semplici e osservino delle regole che permettano loro di cooperare.

- I programmi di una pipeline girano contemporaneamente (non in sequenza) e ciascuno di loro legge il suo Standard Input man mano che i caratteri sono disponibili. E' il kernel che si occupa di sincronizzarli. Cio' implica che nelle pipelines possono esistere programmi interattivi.

- Nessuno dei programmi di una pipeline si rende conto della ridirezione implicita nel meccanismo di pipe. E' la **Shell** che si occupa di disporre le cose in maniera opportuna.

- Non ci sono limiti al numero di comandi in una pipeline e con un po' di fantasia si possono ottenere risultati veramente utili. Per esempio il comando che segue ottiene (in un formato adatto alla stampa) l'elenco dei files appartenenti a root nella directory /tmp ordinati per dimensione (dal piu' grande) e lo memorizza nel file toremove:

```
ls -l /tmp | grep root | sort +4nr | pr > toremove
```

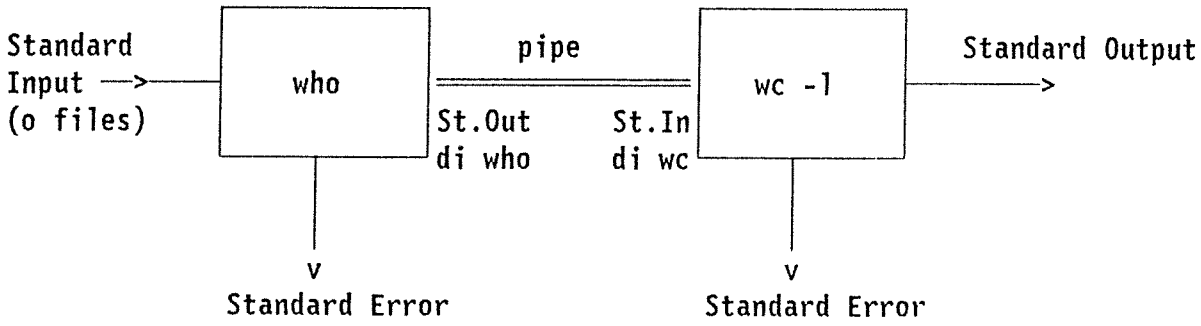
e questa e' una parte del suo output:

```
Thu Apr 18 11:34:45 1991   Page 1


-rw-r--r--   1 root     system    893354 Apr 16 01:21 tangbck.log
-rw-r--r--   1 root     system     94901 Apr 18 11:09 sysdump.log
-rw-rw-rw-   1 root     system      6935 Apr 17 11:48 phase2.out
drwx------   2 root     system       512 Aug 01 1990  lost+found
-rw-rw-rw-   1 root     system       117 Apr 18 11:22 rc.net.out
-rw-r--r--   1 root     system         0 Apr 18 09:59 7d010133.pag
```

- In una semplice immagine il meccanismo di pipe:

```
who | wc -l
```



Processi

- AIX e' un sistema operativo multi-tasking: questo significa che e' in grado di eseguire contemporaneamente piu' programmi. Nella terminologia UNIX ogni programma viene definito **processo**. Un processo e' un programma in esecuzione con uno specifico compito da svolgere.

- Ogni volta che si chiede alla Shell di eseguire un comando (con certe eccezioni) il comando conduce al caricamento in memoria di un programma e alla creazione di un processo.

- Un modo per eseguire due comandi in sequenza e quello di usare il punto-e-virgola ";":

```
$ date; who
Thu Apr 18 12:12:01 CET 1991
peppe       hft/0       Apr 18 10:16
frank       pts/0       Apr 18 10:01
tarricon    pts/1       Apr 18 10:01
peppe       pts/4       Apr 18 10:17
ferretti    pts/3       Apr 18 10:04
$
```

- Ma e' anche possibile avere piu' programmi che girano contemporaneamente. E' ancora la shell che interpreta un metacarattere ed esegue questo compito per noi. Se per esempio volessimo trovare tutti i files che appartengono all'utente test in /u e ordinarli per dimensione, cio' potrebbe condurre ad una attesa piuttosto lunga. Usando il carattere "**&**" possiamo ottenere il prompt della shell prima che il comando abbia terminato la sua esecuzione.

```
$ ls -l `find /u -user test -type f -print 2>error` | sort +4nr >el &
24402
$
```

- Il comando andra' in esecuzione, mentre noi saremo liberi di di svolgere qualche altro lavoro. Il numero

stampato dalla shell e' il **process id** e puo' essere usato per riferirsi allo specifico processo. Se una pipeline (come nell'esempio precedente) viene lanciata in **background**, solo il process-id dell'ultimo processo viene stampato.

- E' importante distinguere tra processo e programma. Ogni volta che il programma **find** viene lanciato in esecuzione, cio' crea un nuovo processo. Possono esistere diverse copie dello stesso programma che girano simultaneamente nel sistema.

- Il comando **wait** attende che **tutti** i processi lanciati con **&** abbiano terminato l'esecuzione.

- Si puo' ottenere l'elenco dei processi che sono in esecuzione con il comando **ps** (process status):

```
$
$ ps
        PID  STAT   TTY  TIME COMMAND
      14166   R   pts/8  0:00 ps
      19920   S   pts/8  0:00 sh
      23891   S   pts/8  0:00 sh
      24402   S   pts/8  0:00 sort
      24660   R   pts/8  0:01 find
$
```

PID e' il process-id, **TTY** e' il terminale ad esso associato; **TIME** e' il tempo di CPU usato (in minuti e secondi).

- Tutti i processi di proprieta' di un utente possono essere terminati con il comando **kill**. Per esempio il comando **kill 24660** terminerebbe l'esecuzione del comando "find" nell'esempio precedente.

- I processi possiedono una struttura gerarchica, simile a quella posseduta dai files nel file system. I processi che create per mezzo dei comandi sono **figli** della Shell che a sua volta e' stata creata da un processo associato alla linea di comunicazione che connette il vostro terminale al sistema (al momento del login).

- Tutti i processi creati con & dalla Shell vengono terminate nel momento in cui fate logout dal sistema (con ctl-D o exit). Il comando **nohup** (no hangup) impedisce che cio' accada. Per esempio **nohup wc -l * &** contera' le linee del vostro programma mentre siete logout e vi fara' trovare l'output nel file **nohup.out** quando farete di nuovo login.

- Ma cosa sono realmente i comandi? Bene quasi tutti i comandi fanno riferimento a **programmi** che sono memorizzati in normali files. Se date una occhiata alle directories **/bin** e **/usr/bin** probabilmente riconoscerete nei nomi dei files alcuni dei comandi che abbiamo usato fino ad ora.

- Quando si immette un comando la shell esegue una ricerca nelle directories elencate nella variabile di **environment PATH**. Una volta trovato il file corrispondente chiede al kernel di iniziarne l'esecuzione, creando cosi' un nuovo processo che sara' figlio della Shell stessa.

- Diamo un'occhiata al PATH del nostro utente test:

```
$
$ echo $PATH
/bin:/usr/bin:/etc:/usr/ucb:/usr/bin/X11
$
```

- Riepiloghiamo in una tabella alcuni dei metacaratteri riconosciuti dalla shell:

Table 2. Metacaratteri

Metacarattere	**Funzione**
>	*prog* >*file* dirige lo Standard Output su file
>>	*prog* >>*file* appende lo Standard Output su file
<	*prog* <*file* Standard Input da file
|	*p1*|*p2* pipe tra p1 e p2
*	qualsiasi stringa di zero o piu' caratteri in filenames
?	qualsiasi singolo carattere in filenames
[ccc]	insieme di caratteri come in [a-z] o [12-4]
;	terminatore di comando: *p1;p2*
`...`	l'output del programma passato come argomento
\	\c interpreta il carattere c alla lettera
'...'	interpreta il comando alla lettera
"..."	interpreta il comando alla lettera tranne per $, `, \
&	lancia il comando in background
$var	valore della variabile di ambiente var

Personalizzazione dell'Ambiente

- E' possibile personalizzare la shell per adattarla alle proprie particolari esigenze. Molte caratteristiche possono essere variate come il **prompt**, il **PATH** di ricerca dei comandi e cosi' via.

- Il comando **stty** permette di modificare il significato di alcuni tasti del terminale (interpretati dalla shell). Per esempio **stty erase '^h'** assegna il carattere di cancellazione al backspace.

- Potete assegnare un nuovo *search path* con il comando **PATH=.:/bin:/usr/bin: ....**

- La variabile di ambiente **PS1** definisce la stringa di prompt. Cosi' **PS1='Ready '** modifichera' il prompt.

- La variabile di ambiente **TERM** permette di adattarsi al proprio modello di terminale.

- E' possibile definire proprie variabili di ambiente per eseguire particolari compiti. Supponete di dovervi spostare spesso tra /u/test/source e /u/test/doc.

```
$
$ source=/u/test/source
$ doc=/u/test/doc
$
$ cd $source; pwd
/u/test/source
$
$ cd $doc; pwd
/u/test/doc
$
```

- Una volta definito l'ambiente opportuno e' possibile memorizzare la sequenza di comandi che lo crea in un file che la shell eseguira' al momento del login.

- Per la **Bourne** shell (che e' quella che abbiamo usato fino ad ora) e per la **Korn** (la shell di default per AIX 3.1) il file si chiama **.profile**

- (Notate che ls visualizza i files che iniziano con "." solo se fornite l'opzione **-a**).

L'editor vi

- Una delle funzioni piu' importanti in ogni sistema operativo e' quella di **editing**. Sapere creare e modificare files e forse il passo piu' importante nell'interazione con un nuovo ambiente di calcolo.

- AIX mette a disposizione diversi **editors**. Sono presenti:
  - ed : editor di linea di cui abbiamo gia' parlato.
  - ex : ancora un editor di linea, forse piu' potente.
  - INed : editor full screen presente su tutti i sistemi AIX prodotti dalla IBM.
  - vi : the **vi**sual editor, forse l'editor full screen piu' conosciuto in ambiente UNIX.

- Due definizioni forse gia' note:

- Gli **editor di linea** permettono di lavorare su una linea alla volta di un file. Sono nati per poter funzionare su terminali di tipo telescrivente.
- Gli **editor full screen** permettono di aprire una **finestra** sul file di lavoro, sono orientati a terminali video con schermo a cursore indirizzabile e permettono di eseguire modifiche spostando il cursore del video sulla parte interessata della finestra.

- **ed** e' nato insieme a UNIX ed e' presente in tutte le piattaforme UNIX (un po' come il comando ls).

- Sia **ex** che **vi** sono stati scritti da William Joy presso la University of California a Berkeley e sono due aspetti dello stesso editor. vi fornisce una interfaccia full screen ad ex, ma i comandi di ex sono tutti disponibili e si comportano nello stesso modo. Lo stesso ex e' basato sul piu' semplice ed, ma e' dotato di estensioni e di caratteristiche addizionali.

- Vedremo alcuni dei comandi base di **vi**. Una premessa: per poter usare un editor di video e' ovviamente necessario disporre di un terminale

video a cursore indirizzabile, ma occorre anche che il sistema sia in grado di gestirlo correttamente. Perche' cio' sia possibile la variabile di environment **TERM** deve essere definita correttamente e il sistema deve possedere la definizione delle sequenze di caratteri necessari ad indirizzare il cursore.

- Studieremo solo i comandi piu' semplici di vi. Uno studio approfondito richiederebbe un intero libro (vi rimando al testo della L. Lamb in bibliografia).

Invocare vi

- Per lanciare **vi** occorre immettere il comando:

```
$ vi [ filename ]
```

Il nome del file puo' essere omesso. In questo caso vi opera su un file il cui nome verra' specificato al momento opportuno.

- Se il comportamento della tastiera non e' quello atteso oppure si ottiene un messaggio di questo tipo:

```
$ vi
Visual needs addressible cursor or upline capability
:
```

significa che la variabile **TERM** non e' definita correttamente o che il terminale su cui state lavorando non ha la capacita' di supportare vi.

- Lanciamo vi in editing sul file "nuovo":

```
$ vi nuovo
```

Lo schermo verra cancellato e sostituito da:

```
~
~
~
~
~
~
...
"nuovo" [New file]
```

- I caratteri "~" (tilde) nella prima colonna a sinistra dello schermo indicano che non esiste testo nel buffer interno di vi, neppure come linee vuote.

- La linea di messaggi nell'ultima riga dello schermo indica il nome e lo stato del file (in questo caso vi

segnala che stiamo creando un file che non esisteva ancora).

- Immettendo il comando **ZZ** usciamo dall'editor e salviamo il contenuto del buffer interno (che e' solo temporaneo) nel file **nuovo** (se non avevamo indicato un nome vi lo segnalera').

- Tentiamo ora di editare un file che esisteva gia':

```
$ vi test
```

```
Questo e' il file test ...
~
~
~
~
~
~
...
"test" 1 line, 27 characters
```

Come potete notare il file viene visualizzato ed ancora le linee vuote vengono segnalate. Il messaggio di stato indica che il file esisteva gia' e la sua lunghezza.

- Il carattere ":" (due-punti) indica a vi di accettare i comandi di ex. Sulla linea di stato sara possibile uitilizzare i comandi dell'editor di linea **ex**.

  Se vogliamo abbandonare il file senza salvare le modifiche possiamo provare con:

```
:q! (RETURN)
```

  Per rileggere il file nel buffer interno (senza salvare le modifiche):

```
:e! (RETURN)
```

- Per salvare il buffer interno potete anche usare il comando ex **w**.

```
:w [filename]
```

  Il comando w (write) puo' essere utile se non si riesce a salvare il file. Se ricevete messaggi del tipo:

```
File exists - use "w! test" to overwrite

File is read only
```

potete usare **w filemame** oppure **w! filename** per forzare vi al salvataggio.

Funzioni di Editing

- Vediamo ora le funzioni fondamentali di editing.

- vi ha due **modi** di funzionamento:
  - Modo **comandi**
  - Modo **inserimento**

- In modo comandi (il modo disponibile alla partenza di vi) l'editor interpreta i caratteri immessi da tastiera come richieste di funzioni sul buffer interno.

- In modo inserimento la tastiera funziona come quella di una normale macchina da scrivere e tutto cio' che immettete viene memorizzato da vi nel suo buffer interno (e visualizzato su video).

- Esistono diversi comandi che indicano a vi di passare in modo inserimento. Uno dei piu' comuni e il comando **i** (insert). Per ritornare al modo comandi e' sufficiente battere il tasto **ESC**.

- Il comando **set showmode** vi permettera' di riconoscere il modo in cui sta funzionando vi. Apparira' il messaggio:

```
INPUT MODE
```

mentre siete in modo inserimento.

- In modo comandi potete posizionare il cursore ovunque nel file. I comandi disponibili sono:

```
                    k    (sopra)

(sinistra)  h                    l  (destra)

                    j    (sotto)
```

e sono (sui terminali che le possiedono) associati ai tasti con le freccette (gli arrows keys). Anche i tasti **RETURN** e **BACKSPACE** svolgono la loro funzione.

- Non e' posibile spostare il cursore oltre una tilde ~ poiche' corrisponde a una linea priva di testo all'interno del buffer.

- Qualunque argomento numerico n premesso ad un comando ne ripete l'esecuzione per n volte: per esempio **4j** sposta il cursore verso il basso di 4 linee (qualunque comando, anche quelli che non si riferiscono al movimento del cursore).

- Altri due utili comandi sono:

```
0 (zero)      va ad inizio linea
$ (dollaro)   va a fine linea
```

- Il comando **i** (insert) passa in modo inserimento ed inserisce il testo dal punto in cui e' posizionato il cursore.

- Il comando **a** (append) passa in modo inserimento ed inserisce il testo dalla posizione che segue il

cursore. (Equivale ad una freccia a destra e al comando i).

- Utili sono anche:

```
w,W   avanti   di una parola  (no punteggiatura)
b,B   indietro di una parola  (no punteggiatura)
```

- Queste semplici sezioni non concludono l'esposizione di vi o di ex. Sono editori abbastanza versatili che richiedono un certo periodo di apprendimento prima di poter raggiungere una certa padronanza nel loro uso.

BIBLIOGRAFIA

Introduzione a UNIX

TITOLO:	The UNIX Programming Environment
AUTORE:	Kernighan, Brian W.
AUTORE:	Pike, Rob
SOGGETTO:	Basic UNIX Programming
EDITORE:	Prentice-Hall
ISBN:	0-13-937699-2, paperback: 0-13-937681-X

TITOLO:	Introducing The UNIX System
AUTORE:	McGilton, Henry
AUTORE:	Morgan, Rachel
SOGGETTO:	Introduction to UNIX
EDITORE:	McGraw-Hill Book Company
DATA:	1983
ISBN:	0-07-045001-3

TITOLO:	Il Sistema Operativo UNIX
AUTORE:	McGilton, Henry
AUTORE:	Morgan, Rachel
SOGGETTO:	Introduzione allo UNIX
EDITORE:	McGraw-Hill Book Company
DATA:	1988 (Edizione Italiana)
ISBN:	0-13-937468-X

TITOLO:	Introducing UNIX System V
AUTORE:	Morgan, Rachel
AUTORES:	McGilton, Henry
SOGGETTO:	Introduction
EDITORE:	McGraw-Hill
DATA:	1987

Shells

TITOLO:	UNIX Shell Programming
AUTORE:	Kochan, Stephen G.
AUTORE:	Wood, Patrick H.
SOGGETTO:	Shell Programming
EDITORE:	Hayden Book Company
DATA:	1985
ISBN:	0-8104-6309-1

TITOLO:	The UNIX C Shell Field Guide
AUTORE:	Anderson, Gail
AUTORE:	Anderson, Paul
SOGGETTO:	C-Shell Guide
EDITORE:	Prentice-Hall
DATA:	1986
ISBN:	0-13-937468-X

TITOLO:	The Kornshell, Command and Programming Language
AUTORE:	Korn, David G.
AUTORE:	Bolsky, Morris I.
SOGGETTO:	Korn shell programming
EDITORE:	Prentice-Hall
DATA:	1989
ISBN:	0-13-516972-0

Editors

TITOLO: Learning the vi Editor
AUTORE: Lamb, Linda
SOGGETTO: Introduction
EDITORE: O'Reilly & Associates, Inc.
DATA: 1988
ISBN: 0-937175-17-X

TITOLO: L'Editor vi
AUTORE: Lamb, Linda
SOGGETTO: Introduzione a vi
EDITORE: Addison-Wesley
DATA: 1990 (Edizione Italiana)
ISBN: 88-7192-009-0

Operating System

TITOLO:	The Design of the Unix Operating System
AUTORE:	Bach, Maurice J.
SOGGETTO:	Design of UNIX
EDITORE:	Prentice-Hall
ISBN:	0-13-201799-7

TITOLO:	UNIX System V Bible
AUTORE:	Prata, Stephen
AUTORE:	Martin, Donald
SOGGETTO:	SYSV Reference
EDITORE:	Howard Sams & Company
ISBN:	0-672-22562-X

Programming

TITOLO:	The AWK Programming Language
AUTORE:	Aho, Al
AUTORE:	Kernighan, Brian
AUTORE:	Weinberger, Peter
SOGGETTO:	AWK Programming
EDITORE:	Addison Wesley
DATA:	1988
ISBN:	0-201-07981-X LC 87-17566

TITOLO:	Advanced UNIX Programming
AUTORE:	Rochkind, Marc J.
SOGGETTO:	Programming
EDITORE:	Prentice-Hall
DATA:	1985
ISBN:	0-13-011818-4, paperback: 0-13-011800-1

TITOLO:	UNIX System V, Complementi di Programmazione
AUTORE:	Thomas, R.
AUTORE:	Rogers, L.R.
AUTORE:	Yates, J.L.
SOGGETTO:	Programmazione in ambiente UNIX
EDITORE:	McGraw-Hill
DATA:	1988
ISBN:	88-386-0608-0

C Programming

TITOLO:	The C Programming Language, First Edition
AUTORE:	Kernigan, Brian W.
AUTORE:	Ritchie, Dennis M.
SOGGETTO:	C Programming
EDITORE:	Prentice-Hall
DATA:	1978
ISBN:	0-13-110163-3

TITOLO:	The C Programming Language, Second Edition
AUTORE:	Kernigan, Brian W.
AUTORE:	Ritchie, Dennis M.
SOGGETTO:	C Programming
EDITORE:	Prentice-Hall
DATA:	1988
ISBN:	0-13-110362-8

TITOLO:	The C Answer Book
AUTORE:	Tondo, Clovis L.
AUTORE:	Gimpel, Scott E.
SOGGETTO:	C programming
EDITORE:	Prentice-Hall
DATA:	1985
ISBN:	0-13-109877-2

Networking

TITOLO:	Internetworking with TCP/IP
AUTORE:	Comer, Douglas
SOGGETTO:	Networking
EDITORE:	Prentice-Hall
DATA:	1988
ISBN:	0-13-470154-2

TITOLO:	UNIX Network Programming
AUTORE:	Stevens, W. Richard
SOGGETTO:	UNIX Networking
EDITORE:	Prentice Hall
DATA:	1990
ISBN:	0-13-949876-1