Istituto Nazionale di Fisica Nucleare **Sezione di Cagliari - CA/NI/3-96**

Mini-Guida

Unix

Giugno,1996

a cura di

Antonio Silvestri

Indice

1. Introduzione 5

1.1 Cenni storici 5
1.2 UNIX o OpenVMS ? Cos'è meglio ? 6
1.3 Account 7
1.4 Login, Logout 7
1.5 Password 8

2. Unix Shells 9

2.1 Stdin, Stdout e Redirection 11
2.2 Pipe e filtri 11
2.3 Shell script 12
2.4 Regular Expression 13
2.5 Editors 14
2.5.1 vi 14
2.5.2 emacs 14
2.5.3 edt 14
2.6 Variabili di Ambiente 14
2.7 I files "Profile" 15
2.8 Terminali 16
2.9 Alias 16
2.10 Richiamo comandi precedenti 17

3. Unix File System 19

3.1 Comandi fondamentali 19
3.1.1 Elenco del contenuto di una directory 19
3.1.2 Cambiamento di directory 19
3.1.3 Mostra la directory corrente 20
3.1.4 Crea una directory 20
3.1.5 Cancella una directory 20
3.1.6 Rinomina/sposta una directory o un file 20
3.1.7 Visualizza il contenuto di un file 20
3.1.8 Copia un file 20
3.1.9 Cancella un file 20
3.2 Controlli di accesso 20
3.3 Backup 21
3.3.1 tar 21
3.3.2 vbackup 22

4. Comunicazioni 23

4.1 Il protocollo TCP/IP 23
4.2 Telnet 23
4.3 Ftp 24

4.4 Altri Servizi 25
4.4.1 rsh 25
4.4.2 rlogin 25
4.4.3 rcp 25
4.5 Stampa 26
4.6 Mail 27
4.7 Pine 28

5. SOFTWARE 29

5.1 Compiling and linking 29
5.2 Librerie 29
5.2.1 Il comando cernlib 30

6. Come fare per … 31

6.1 Cancellare un file il cui nome inizia con “-” ? 31
6.2 Cancellare un file con caratteri “strani” nel nome ? 31
6.3 Inserire il pathname corrente nel prompt ? 31
6.4 Rinominare files “*.xyz” in “*.vwk” ? 32
6.5 Trovare la data di creazione di un file ? 32
6.6 Recuperare un file cancellato con “rm” ? 32
6.7 Eseguire programmi con input da uno script o in background ? 32
6.8 Riportare la data in un nome di file ? 33
6.9 Come scambiare dati con nastro tra UNIX e OpenVMS ? 33
6.9.1 tar 33
6.9.2 ltf 33
6.10 Trovare il process ID di in programma ? 34

Appendice A: UNIX per Esempi 35

Appendice B: Corrispondenza tra comandi UNIX e OpenVms 43

Bibliografia 45

Capitolo 1

1. Introduzione

Lo scopo di questa piccola guida è quella di fornire le prime basi del Sistema Operativo Unix. Essa si rivolge a tutti coloro che non lo conoscono e desiderano averne un quadro generale.

Sebbene sin dall'inizio l'INFN abbia usato principalmente l'OpenVms la situazione attuale è cambiata. Infatti l'utilizzo di OpenVms (al quale faremo riferimento anche col nome breve VMS) e di Unix è quasi paritario.

1.1 Cenni storici

Unix è un sistema operativo non proprietario molto diffuso nell'ambiente Universitario, nei centri di ricerca e negli enti di sviluppo software. Motivo principale del suo successo è la sua relativa semplicità di installazione e trasporto sulle più diverse piattaforme hardware, anche le più economiche.

Un **Sistema Operativo** è un complesso di programmi che controllano e gestiscono le varie parti di cui un computer è composto, comprese le eventuali periferiche, ed ha inoltre lo scopo di fornire all'utente una serie di servizi per renderne l'uso il più agevole possibile.

La sua storia è per molti aspetti peculiare e di sicuro unica se la confrontiamo con tutti gli altri SO[1] presenti nello scenario informatico mondiale. Unix può essere considerato il successore di **Multics**[2] un Sistema Operativo progettato negli anni 60 dal **MIT**[3], **GENERAL ELECTRIC** e **AT&T Bell Laboratories**. Tale progetto, un mastodonte sia burocratico che tecnologico, non giunse mai a compimento. Ken Thompson e Dennis Ritchie erano dei ricercatori che vi avevano lavorato e non volendo abbandonare completamente un ambiente di programmazione al quale erano abituati e vicino alle loro esigenze, cominciarono ad interessarsi alla realizzazione di un SO snello e flessibile da installare su computers a basso costo. Nel 1969 nacque così UNIX il quale conteneva molte delle caratteristiche di Multics e che nelle intenzioni dei suoi ideatori doveva essere la concretizzazione di idee di semplicità, compattezza, portabilità e indipendenza tecnologica.
In seguito Unix si sposta dall'**AT&T** verso l'ambiente Accademico Statunitense a cominciare dall'Università di **Berkeley** che avrebbe avuto un ruolo attivo nella sua diffusione. Infatti da qui comincia la prima diversificazione tra lo Unix di **AT&T** detto anche **System V** e quello di Berkeley chiamato **BSD** (Berkeley Software Distribution).

[1] Abbreviazione di Sistema Operativo

[2] **Mult**iplexed **I**nformation and **C**omputing **S**ervice

[3] **M**assachusetts **I**nstitute of **T**echnology

Da questo momento Unix si diffonde velocemente nel settore della Ricerca Scientifica Universitaria ed in quello del privato con una grande proliferazione di sistemi "Unix" derivati da System V o da BSD (o da una mistura dei due). Attualmente ci troviamo nella situazione di avere tanti SO Unix quanti sono le piattaforme hardware esistenti; questi, sebbene siano relativamente simili dal punto di vista dell'utente generico, possono invece risultare molto diversi nella parte che riguarda il System Management.

Unix è stato oggetto di molte critiche per avere in gran parte tradito quelle caratteristiche per il quale era nato, giudicate da alcuni illusorie e quindi solo dei miti. Molti lo hanno definito un "Un-Operating System" e lo hanno accusato di essere un chiaro esempio che *essere compatti e semplici è più importante che essere completi e corretti*. Qualcuno è arrivato ad affermare che la sua diffusione è stata una iattura per l'informatica avendo, con la sua esistenza, limitato lo sviluppo verso SO più efficienti e sicuri.

1.2 UNIX o OpenVMS ? Cos'è meglio ?

Questa è una di quelle domande periodiche che vengono poste da molti utenti. Rispondere è molto semplice a patto che si mettano da parte fanatismi e preconcetti. In generale si può affermare che l'OpenVms e le migliori implementazioni di Unix sono buoni SO, ognuno con i propri punti deboli e punti di forza . Se siete nella posizione di poter scegliere date la preferenza a quello che meglio soddisfa la vostre esigenze, considerando, per esempio, se un prodotto software o una particolare caratteristica del SO che ritenete utile (o indispensabile) è disponibile o meno.

Allo scopo di dare un'idea delle caretteristiche principali di questi due SO, senza entrare nei dettagli, ecco alcuni dei pregi e dei difetti posseduti da ciascuno.

Vantaggi Unix: E' un Sistema Operativo non proprietario (quasi gratis); è cresciuto nell'ambito Universitario degli Stati Uniti d'America ove ha prosperato (molto) liberamente. La manipolazione dei dati e dei files è semplice; quasi tutto è riconducibile a files ed ogni input è una sequenza di byte (stream); questo ha il vantaggio di poter collegare Unix ad qualsiasi tipo di *file system*. Vanta un alto livello di integrazione con i protocolli TCP/IP i quali sono così diventati uno standard "de facto". E' molto diffuso ed esiste un notevole parco software fornito da inummerevoli aziende del settore ed una quantità enorme di software di Pubblico Dominio. Unix è molto flessibile ed è dotato di una elevata configurabilità grazie alla quale è possibile fare cose molto complesse.

Svantaggi Unix: Interfaccia utente pessima; comandi oscuri e criptici (alcuni potenzialmente pericolosi). Questo è principalmente dovuto all'obsolescenza dell'architettura del SO. La gestione del sistema è macchinosa e lastricata di trappole; non esiste un unico approccio alla gestione del sistema ma uno per ogni piattaforma hardware (ogni produttore ha introdotto varianti sia per addatarlo alla propria piattaforma sia per avvicinarlo alle nuove tecnologie). La gestione delle risorse è pesante ed inefficiente. Il file system (ufs) non è molto robusto; non è raro imbattersi in files corrotti. Ma forse il difetto più grave è la sicurezza; da questo punto di vista Unix è un vero e proprio incubo per il gestore.

Vantaggi Vms: Una accettabile ed amichevole interfaccia utente con comandi generalmente auotoesplicativi. La gestione del sistema è abbastanza semplice:

esistono, di solito molte procedure standard per effettuare tutte le principali attività di "system management". La gestione delle risorse è semplice; il file system è robusto. La sicurezza dei sistemi VMS è ottima: se si seguono le regole suggerite dai manuali è molto difficile subire un intrusione nel proprio sistema. Possiede un sistema di code *batch* (esecuzione/stampa) molto sofisticato e totalmente integrato nel SO, ed un invidiabile livello di condivisione delle risorse (tra più macchine anche a distanze geografiche) con la tecnologia *cluster*. Un altra caratteristica del VMS è quella di essere in continua evoluzione, avvantaggiandosi efficacemente delle nuove tecnologie.

Svantaggi Vms: E' un SO proprietario legato ad un solo distributore e quindi con tutti i problemi che da questo derivano. Questa situazione, comunque, è cambiata negli ultimi anni con un'ottima politica di "apertura" verso l'esterno (Esempio: accordi di integrazione OpenVms/Microsoft). Naturalmente è meno diffuso dello Unix e di conseguenza esiste meno Software disponibile. Il protocollo di comunicazione usato dai sistemi DEC è, in genere, il *Decnet*. Per implemetare la suite dei protocolli TCP/IP bisogna acquistare dei package a parte.

1.3 Account

Per accedere ai computers con il SO UNIX ogni utente deve possedere un "account", il quale consiste nella registrazione di un profilo (nome, cognome, directory di lavoro, tipo di utente ecc.) e nell'assegnazione di un *username* (nome utente) e di una *password* (parola segreta o parola d'ordine) iniziale. Queste operazioni sono generalmente effettuate dal proprietario della macchina oppure, nel caso di sistemi con gestione centralizzata, dal Responsabile del Centro di Calcolo. Lo *username* è il nome con il quale l'utente è conosciuto dal SO ed insieme alla password permette di venire riconosciuto come utente autorizzato.

ATTENZIONE Unix, a differenza di altri SO, come per esempio l'OpenVMS, è sensibile ai caratteri maiuscoli e minuscoli; quindi, due comandi, due nomi di file, due opzioni o due parametri che differiscano anche di un solo carattere, maiuscolo in uno minuscolo nell'altro, sono considerati diversi.

1.4 Login, Logout

Una volta in possesso di un account su una determinata macchina potete fare *login*[4]. Questo è il termine con il quale si indica la sequenza delle operazioni che permettono ad un'utente di collegarsi con un computer col SO Unix. Durante questo stadio avviene il suo riconoscimento e la creazione di un "processo" nell'ambito del quale egli potrà, poi, svolgere le sue attività.
Se siete davanti ad un **terminale**, troverete la dicitura "login:" che vi invita a digitare il vostro *username* (seguito dal tasto <RETURN>; d'ora in poi sarà impicito il suo uso alla fine di ogni tipo di input da tastiera); subito dopo compare la scritta "password:" dopo la quale dovete digitare la password (che resterà invisibile) seguiranno dei messaggi più o meno lunghi ed infine apparirà il simbolo "#" (detto *prompt*) che indica che il sistema è pronto a ricevere comandi. Il simbolo usato come prompt può

[4] registrarsi, identificarsi. **log:** nota scritta ufficiale che descrive un evento, un esempio è il "giornale di bordo" di una nave; **to log:** significa fare questo tipo di registrazione.

anche essere diverso perchè è legato al tipo di ambiente di lavoro (shell) nel quale vi trovate.
Nel caso il vostro terminale sia collegato ad un **Terminal Server** (di solito Digital), noterete il prompt "`Local>`" sullo schermo; per collegarvi al vostro computer Unix preferito dovete digitare

`Local>`**`connect`** ***`servizio`*** (oppure ***`c servizio`***)

o semplicemente

`Local>` **`c`**

questa forma è valida solo se sul Terminale Server è stato definito un servizio "preferito". Il termine "servizio" è qui inteso come sinonimo di computer (or *host*) al quale è possibile connettersi. Il comando

`Local>`**`show services`**

vi fornisce un elenco dei possibili *servizi*.
Il *login* su una **workstation** Unix è simile; infatti, dopo aver inserito lo *username* e la *password* nella *login window*, a secondo del costruttore possono comparire varie finestre con le quali l'utente interagisce con il sistema. Nel caso delle workstation Unix della DEC compare la finestra del "Session Manager" con un menu lineare. Selezionando con il mouse la voce "Applications" appare un "pull-down menu" con numerose applicazioni; tenendo, quindi, sempre premuto il pulsante del *mouse,* e muovendo il *pointer* in corrispondenza della voce "Decterm" e rilasciando il pulsante, dopo qualche secondo appare una finestra simile a quella di un terminale con il prompt "#" (come si è già detto questo può variare).
Quando volete chiudere la sessione (**logout**) con un sistema Unix digitate "logout" se vi trovate in una C-shell , "exit" se siete in una Bourne shell. Il carattere di controllo <Ctrl-D> (chiamato carattere di End Of File, quasi l'equivalente del <Ctrl-Z> in OpenVMS) realizza la stessa funzione con entrambe le shell. Se avete processi sospesi in background (non interattivi) verrete avvisati della loro esistenza e dovrete ripetere il logout per ognuno di essi. Nel caso di una Workstation di solito esiste un menu con una voce speciale che indica la chiusura della sessione. In DEC Unix, per esempio, nella finestra del "Session Manager" c'è la voce "Session" che attiva un pull-down menu contenente l'azione "End Session".

1.5 Password

Per cambiare la password (la lunghezza minima a secondo dell versioni varia da 4 a 6 caratteri) usate il comando

`passwd`

questo vi chiederà la vostra password attuale e successivamente due copie della nuova password per verifica. Si consiglia di evitare l'uso di password ovvie.
Si faccia attenzione che Unix (senza estensioni) prende in considerazione solo i primi 8 caratteri ignorando del tutto gli altri e distingue le maiuscole dalle minuscole.

Capitolo 2

2. Unix Shells

Una volta che avete fatto login vi trovate in un ambiente chiamato **shell**. Questo è un processo che viene creato alla fine della sequenza di login. Una shell non è altro che l'interfaccia tra l'utente ed il SO (esattamente come il DCL per l'OpenVMS).
Esistono due famiglie di shell: la prima è basata sulla Bourne Shell (**sh**), la seconda sulla C-Shell (**csh**). La Korn Shell **(ksh**), la Bourne Again Shell (**bash**) e la superKorn (**zsh**) shell fanno parte della famiglia sh. La **tcsh** è un miglioramento della csh.

L'uso di una shell piuttosto che un altra è una scelta del tutto personale ed è legato al tipo di attività che si vuole intraprendere. Per sapere qual è la vostra shell di default digitate

```
printenv (o env)
```

il quale elenca tutte le variabili di ambiente, tra le quali noterete la variabile SHELL, il cui valore è il pathname del programma shell (per esempio: SHELL = /bin/csh). Nel caso vogliate cambiare il tipo di shell di default del vostro account potete farlo con il comando

```
chsh
Changing login shell for gianni
old shell: /bin/csh
new shell:/bin/ksh
```

nell'esempio sovrastante un utente ha cambiato la propria shell di default da **csh** a **ksh**.

La struttura sintattica dei comandi Unix è la seguente:

```
command [-option(s)]  [argument(s)]
```

command rappresenta il nome del programma o dell'utility; le **option** (precedute da un trattino e separate da spazi) sono dei sottocomandi che modificano l'azione del **command**; gli **arguments** sono i dati sulle quali agisce il **command** (se più di uno sono separati da spazi).

Se volete dare più di un comando su una stessa riga li dovete separare con un "**;**". Esempi:

```
ls; cat prog.c
cc -o file_di_output prog.c
```

Se volete spezzare (per qualsiasi motivo) la linea di comando su più righe usate il simbolo "\". Esempio:

```
cat aa.txt \
       prog.c
```

Per ottenere maggiori dettagli sui comandi Unix potete utilizzare le **man pages**, in pratica si tratta di veri e propri manuali consultabili da terminale che descrivono tutti (o quasi) i comandi Unix. Se volete, per esempio, i dettagli sul comando "ls" digitate

```
man ls
```

e sul terminale appariranno tutte le informazioni sul comando "ls".

Quando digitate un comando, la shell verifica se è un programma interno (built-in) o esterno alla shell. Nel primo caso il comando viene eseguito immediatamente; nel secondo viene ricercato un file, con lo stesso nome, in tutte le directory specificate dalla variabile di ambiente PATH (*search path*). Se il file non è trovato in queste directory, la shell vi avvisa che il comando non esiste.
Se state usando la csh o la tcsh e aggiungete un comando (un file) in una directory compresa nel vostro *search path*, dovete lanciare il programma **rehash** (o fare un logout seguito da un login)**;** in modo che siano ricreate le tabelle interne alla shell. In caso contrario sarete avvisati che il file non esiste.

La shell nel quale vi trovate, dopo il login, è *child process* di un *parent process* creato dal SO al termine della sequenza di login. Creare un nuovo processo a partire da un altro viene chiamato, in ambiente Unix, *forking*. Questo nuovo processo viene detto *child process*, il quale a sua volta può diventare il *parent process* di un altro. Un processo, sia esso sospeso (stopped) o in esecuzione (running), non associato ad alcun terminale viene definito processo di *background*. Per avere informazioni sui vostri processi potete usare il comando

```
ps
```

```
PID   TTY      STAT   TIME    COMMAND
1303  lat/620   S     0:00.25  -csh (csh)
2273  lat/620   R     0:00.02  ps
```

La risposta descrive i seguenti parametri: PID (Process Identifier) è il numero che identifica il processo; TTY è il terminal e associato al processo; TIME è il tempo di CPU fino a quel momento usato e COMMAND il comando in esecuzione. La colonna STAT fornisce dati sullo stato del processo. Il significato dei principali codici di stato è il seguente:

Running	in esecuzione
Sleeping	in attesa (per meno di 20 secondi)
Idle	in attesa (per più di 20 secondi)
T	stopped
Uinterruptible	processo in attesa che non può essere interrotto
Zombie	processo non terminato correttamente
s**W**apped	cioè tolto dalla memoria centrale e trasferito su disco

Per ottenere un output più dettagliato usate il comando

```
ps xu
```

```
USER   PID  %CPU  %MEM  VSZ   RSS  TTY  STAT  STARTED  TIME     COMMAND
silv   543   0.1   0.2   1.0M  10K  ??    R    Jun 17   0:00.10  sort
```

Le informazioni in più riguardano l'utente (USER), uso della CPU (%CPU), uso della memoria centrale (%MEM), uso della memoria virtuale (VSZ) ed occupazione totale di memoria centrale (RSS).

Si avverte che i comandi Unix sono noti per essere poco standardizzati e dal punto di vista mnemonico poco utili data la totale assenza di regole. Infatti, succede spesso che i comandi (e non solo questi), insieme alle loro opzioni, ai loro argomenti ed all'effetto da essi prodotto siano suscettibili di variazioni (anche notevoli) da una piattaforma Unix all'altra. Quindi non è detto che tutto quello che è descritto nella presente guida sia sempre valido.

2.1 Stdin, Stdout e Redirection

La shell, ed in generale tutti i programmi Unix, prendono l'input dal cosiddetto *standard input* (**stdin**) e scrivono sullo *standard output* (**stdout**). Esiste anche un output riservato agli errori chiamato *standard error* (**stderr**). Per default lo standard input è associato alla tastiera e lo standard output e lo standard error allo schermo del terminale.
Redirection (ri-indirizzo) significa ridefinire queste assegnazioni con l'ausilio di particolari caratteri. La struttura di una riga di comando con redirection è la seguente:

```
cmd [-option(s)] [argument(s)] < input.file >output.file
```

cioè *cmd* prende l'input da *input.file* e scrive l'output su *output.file*. Ecco una lista di caratteri che espletano questa funzione sono:

>	riindirizza lo standard output
>**&**	riindirizza lo standard error
<	riindirizza lo standard input
>>	aggiunge allo standard output (cioè scrive in coda ad un file esistente)
>>**&**	aggiunge allo standard error

Esempi:

```
ls -l > elenco                  associa stdout al file elenco
(cat myfile > outF)>& outE      associa stdout e stderr al file diversi
(who; ls ) > elenco.txt         associa stdout di entrambi i comandi
who; ls > elenco                associa stdout solo di ls
```

2.2 Pipe e filtri

Il concetto di *pipe*[5] è uno delle caratteristiche di Unix utilissime all'utente. Il suo scopo è quello di connettere lo standard output di un programma allo standard input di un altro. L'operatore di pipe è la barra verticale "|". Esempio:

```
ls | sort
```

[5] condotto, tubo.

l'output del comando ls viene passato come input al comando sort, che lo mette in ordine alfabetico (rispetto alla prima colonna) e lo passa a stdout (il terminale, se non è stato rifefinito). L'output di un comando può essere duplicato e passato come input a due comandi diversi con **tee**[6].

```
ls | tee elenco| sort
```

l'output di ls viene, sia scritto nel file "elenco" che passato come input al programma sort.

Un *filtro* è un programma (**grep**[7], **sort**, **awk**[8], **paste**, **cut**, **pr**) che manipola l'input secondo certe regole e crea un output consistente con queste ultime. La combinazione delle pipe e dei filtri forniscono un mezzo molto potente per la manipolazioni dei dati.

Per esempio **awk** è una potente utility per l'elaborazione di testi. E' in grado di fare operazioni complesse che di solito sono affidate a linguaggi di programmazione come lo C o il Fortran. Di questo programma ne esiste una versione più recente chiamata **nawk**; tuttavia, al suo posto molti preferiscono usare l'utility **perl**[9] di Larry Wall.

2.3 Shell script

Un "Shell script" è l'equivalente Unix di una procedura DCL in OpenVms o di un file BAT in MS-DOS. In altre parole si tratta di un file di comandi che viene interpretato dalle shells Unix. In genere gli scripts sono scritti per la Bourne Shell sia perché si dice essere più adatta a questo scopo sia perché è la shell di default.

Affinché uno script possa essere eseguito il file che lo contiene deve essere abilitato all'esecuzione (permesso o privilegio di esecuzione).

```
chmod +x filecommand      abilita l'esecuzione
filecommand               esegue il file di comandi
```

E' importante tenere presente che, come tutti i comandi che non sono intrinseci alla shell, un file script è eseguito nell'ambito di un processo *child* creato (forked) dalla shell *parent*. Questo significa che la shell child eredita tutte le caratteristiche (variabili di ambiente) della parent shell. Tuttavia alla fine dell'esecuzione dello script, il controllo ritorna alla parent shell e qualsiasi definizione fatta dal processo child viene persa. Se si vuole che uno script modifichi le definizioni della shell corrente bisogna usare i comandi:

```
source myscript           per la C-shell
. myscript                per la Bourne-shell
```

Si noti lo spazio dopo il punto.

Per imparare a scrivere *shell script* consultate alcuni titoli sull'argomento presenti nella bibliografia alla fine di questa guida.

---

[6] giunzione a T (di un tubo)

[7] **G**lobal **R**egular **E**xpression **P**rint

[8] **A**ho **W**einberger and **K**ernigham, gli autori del linguaggio.

[9] **Pr**actical **E**xtraction and **R**eport **L**anguage (oppure **P**athologically **E**cletic **R**ubbish **L**ister)

2.4 Regular Expression

Una *regular expression* è una stringa composta da lettere, numeri e simboli (detti anche metacaratteri) la quale, interpretata da un programma (di solito ed, grep, awk, ecc.) descrive un certo risultato. Per esempio l'espressione "`E.E`" seleziona (match) le stringhe `AERE0`, `PRETESA`; infatti in questo caso il significato dell'espressione "`E.E`" è quello di selezionare qualsiasi stringa che possegga le due lettere "E" con un altro carattere qualsiasi tra di esse interposto.
Di seguito si fornisce una tabella dei simboli che possono essere usati nella costruzione delle *regular expression.*

Notazione	Significato	Esempio	Match
.	un carattere singolo qualsiasi	t.	*t* seguito da un carattere qualsiasi
^	individua l'inizio di una riga	^The	i caratteri *The* solo se sono all'inizio della riga
$	fine della riga (End Of Line)	x$	*x* solo solo se si trova alla fine di una riga.
		^and$	Una riga che contiene solo *and.*
		^$	Una riga che non contiene caratteri.
*	nessuna o più ripetizioni dell'espressione precedente	y*	Nessuno o più ripetizioni consecutive di y
		yy*	Una o più ripetizioni consecutive di y
		.*	Nessuno o più caratteri (qualsiasi) consecutivi
		\.*	Tutte le righe che terminano con un punto. Si noti l'uso del *backslash* "\" come carattere di *escape* per evitare che il punto venga interpretato come metacarattere.
		a.*b	Tutti i caratteri compresi tra a e b.
[*car*]	qualsiasi carattere o intervallo di caratteri presente nella stringa *car*	[tT] [a-z] [A-Za-z]	sia t che T qualsiasi carattere minuscolo qualsiasi carattere sia minuscolo che maiuscolo
[^*car*]	qualsiasi carattere NON presente in *car*	[^0-9]	qualsiasi carattere non numerico
\{ *min*,*max*\}	un minimo di *min* fino ad un massimo di *max* ricorrenze della precedente espressione	x\{1,5}\ [0-9]\{3}\ [0-9]\{3,}\	ogni sequenza di 1 fino 5 caratteri *x* Solo 3 cifre almeno 3 cifre

2.5 Editors

2.5.1 vi

Questo editor ha la stessa età del SO Unix e quindi estremamente poco moderno; tutti lo possono usare digitando "vi" al prompt della shell[10].

2.5.2 emacs

Emacs è un potente editor molto popolare nel mondo Unix. Esso appartiene al progetto GNU del Free Software Foundation ed e disponibile su quasi tutte le piattaforme hardware. In effetti emacs è più di un editor è quello che si dice un ambiente di lavoro completo, si può fare tutto dal suo interno senza mai uscirne.

2.5.3 edt

Edt è un programma che fornisce tutte le funzionalità dell'editor EDT dell'OpenVms ed è quindi molto utile a quanti non vogliono soffrire di bruschi cambiamenti di ambiente di lavoro.

2.6 Variabili di Ambiente

Nelle shell esistono numerose variabili (Environment Variable e shell variable) che definiscono un certo numero di parametri i quali vengono messi a disposizione dei programmi. Alcuni di queste variabili sono già predefinite dal SO; l'utente, a sua volta, può definirne altre o modificare quelle già esistenti. Per ottenere una lista delle variabili ambientali digitate:

```
env             Bourne Shell
printenv        C-shell
```

Per esempio per definire o modificare la variabile di ambiente PRINTER date i seguenti comandi:

```
setenv PRINTER nome_stampante            C-shell
PRINTER=nome_stampante; export PRINTER   Bourne shell
```

Se il valore da assegnare alla variabile contiene degli spazi dovrete includerlo tra una coppia di doppi apici. Per verificare il valore di una singola variabile digitate il comando

```
echo $PRINTER
tps20
```

Si noti il simbolo del "$" ha il significato di "mostra il valore della variabile che segue". Se volete modificare il simbolo del prompt dovrete cambiare il valore delle seguenti variabili (di shell):

```
PS1="Silv"                Bourne shell
set prompt="Silv"         C-shell
```

[10] Si declina ogni responsabilità sulle eventuali turbe psicologiche derivanti dall'uso prolungato di tale programma.

Volete che il prompt riporti il percorso della directory nel quale vi trovate ? Allora date il comando:

```
set prompt="$cwd>"
```

Naturalmente vi sono un gran numero di altre variabili di questo tipo; ecco qui un elenco delle più importanti:

HOME	definisce la directory di default subito dopo la sequenza di login
PRINTER	definisce la stampante di default
PATH	definisce i percorsi di ricerca che la shell deve consultare quando deve cercare un comando
PS1 *(Bourne shell)* prompt *(C-shell)*	definisce il simbolo del prompt
EDITOR	definisce l'editor di default
TERM	Il tipo di terminale previsto come output
DISPLAY	definisce il server X remoto verso cui indirizzare la visualizzazione

2.7 I files "Profile"

Ogni shell in Unix, quando viene attivata, richiama dei file di inizializzazione denominati file di "Startup" o file "Profile". I nomi di questi file, la loro posizione e il loro ordine di esecuzione variano da shell a shell. Segue una tabella esplicativa dei files che vengono eseguiti a secondo della shell e del contesto nella quale quest'ultima viene eseguita.

Shell	Situazione	Script file eseguito (nell'ordine)
csh	ad ogni lancio	$HOME/.cshrc
	al login	/etc/csh.login, $HOME/.cshrc,$HOME/.login
	al logout	$HOME/.logout
sh	al login	/etc/profile, $HOME/.profile
ksh	al login	/etc/profile, $ENV[11],$HOME/.profile
	ad ogni lancio	$ENV
	al logout	qualsiasi comando che sia stato definito con: trap "comando" 0
bash	al login	/etc/profile,$HOME/.bashrc,$ENV, $HOME/.bash_profile (se .bash_profile non è presente viene eseguito $HOME/.profile)
	ad ogni lancio	$HOME/.bashrc, $ENV[12]
	al logout	$HOME/.bash_logout

[11] Il file definito da questa variabile.

Quindi se volete modificare od aggiungere azioni di vostra scelta al momento del login non dovete fare altro che creare o modificare questi files “nascosti”.

Sono definiti nascosti perché iniziando con un “.” sono visibili solo col comando “ls -la” o “ls -a”. Noterete anche nella colonna dei nomi dei file un “.” e “..” questi sono i puntatori rispettivamente alla directory corrente ed alla directory “padre” in modo equivalente a quello che accade nel MSDOS.

2.8 Terminali

Se il terminale di default, descritto dalla variabile TERM, è diverso da quello che avete a disposizione potete cambiarlo con il comando

set term=*termtype* C-shell

dove *termtype*, di solito, può prendere i valori VT100 e VT220. Le funzioni di controllo del terminale possono essere visualizzate con il comando

stty -a

per ulteriori informazioni su stty consultate le man pages con man stty.
Nelle workstation con l’interfaccia X-window per poter ri-indirizzare l’output di un programma su altri computer con server X viene usata la variabile di ambiente DISPLAY, per esempio l’istruzione

setenv DISPLAY calf34.ca.infn.it:0.0 C-shell

re-indirizza l’output di tutti i programmi con interfaccia X sulla workstation calf34. Molti programmi possiedono come opzione l’indirizzamento diretto verso un determinato host, di solito la sintassi è del tipo

program -display calf34.ca.infn.it:0.0

2.9 Alias

Tutte le shell eccetto la Bourne shell (sh) permettono la definizione di comandi diversi da quelli standard con il sistema degli “alias”. In pratica esiste un comando (alias) con il quale potete definire equivalenze tra i comandi Unix e qualsiasi stringa di caratteri di vostra scelta.

```
alias dir ls                    C-shell
alias dir ls -l
alias dirs ’ls -l \!* | more’
alias dird ’ls -l| grep ^d’     (lista solo le directory)
alias dir=ls                    Korn shell
alias dir=’ls -l’
```

[12] Nel caso di una sessione non interattiva viene eseguito solo il file definito da questa variabile.

2.10 Richiamo comandi precedenti

Tutte le shell tranne la Bourne shell possiedono un meccanismo di memorizzazione dei comandi che sono stati eseguiti. Il comando per produrre tale lista è

```
history
```

Nella **C-shell** sono disponibili anche i seguenti comandi:

!!	riesegue il comando precedente
!n	riesegue il comando numero n
!text	riesegue il comando più recente che inizia con la stringa “text”
!?text	riesegue il comando più recente contenente la stringa “text”

La funzione *history* è attivata dal comando “`set history=`*n*” che di solito viene inserito nel file “.cshrc”. *n* è il numero di comandi che si vogliono tenere in memoria.

Nella **Korn-shell** si possono attivare una serie di comandi per mezzo caratteri di controllo per fare del *line editing*. A tale scopo bisogna prima digitare **`set -o emacs`** (è più pratico inserirlo in uno dei file di inizializzazione come. ad esempio, .kshrc). Essi sono:

Ctrl-p	estrai il comando precedente
Ctrl-n	estrai il comando seguente
Ctrl-b	sposta il cursore indietro (nella riga di comando), freccia sinistra
Ctrl-f	sposta il cursore in avanti, freccia destra
Ctrl-d	cancella carattere (il tasto è uguale a quello **delete**)
Ctrl-a	vai all’inizio (della riga di comando)
Ctrl-e	vai alla fine

Per maggiori dettagli consultate le *man pages*.

Capitolo 3

3. Unix File System

Un “file” (o archivio) è un insieme di dati (records) che può essere visto come una singola entità. Una “directory” è un file i cui records sono dei puntatori ad altri files o directory.
Lo Unix, come del resto anche l’OpenVMS, gestisce lo spazio di un disco creando una struttura logica, detta *file system*, che è quella di un albero rovesciato, con “directory” e “sub-directory”. Come regola generale alla base dell’albero abbiamo la **root** (identificata con il simbolo “/”) e sotto di essa le directory **/usr** (directory user), **/bin** e/o **/sbin** (directory dei files binari, cioè tutti i files *eseguibili*), **/dev** (directory dei *special files*, che rappresentano dispositivi hardware e software), **/etc** (directory dei file di sistema) e **/tmp** (directory per la creazione di files temporanei). Tutti gli altri file, compresi quelli degli utenti, si trovano al disotto di queste directory o della root (di solito /usr/users o /users o anche /home).
Un file è quindi rappresentato, ad esempio, dal percorso (o *pathname*):

```
/usr/users/silv/filename
```

`filename` può essere qualsiasi combinazione di caratteri (fino quattordici per alcune versioni di Unix, fino a 255 per altre) eccetto i simboli slash (/), asterisco (*), apici (‘), doppi apici (“”), parentesi quadre ([]), dollaro ($), punto interrogativo(?), backslash (\), Ampersand (&), brackets (<>).

Attenzione questo significa che un nome di file può essere composto anche da caratteri di controllo e quindi invisibili al comando ls.

Esiste anche un’altra categoria di entità che non sono nè file nè directory i **symbolic** e **hard links**. Questi non sono altro che un nome aggiuntivo per uno stesso file. Gli **hard links** creano nomi aggiuntivi di soli files nell’ambito di uno stesso file system. I **symbolic links** non solo possono creare ulteriori nomi per files attraverso file system diversi ma anche per directory.

3.1 Comandi fondamentali

Per una più completa lista di comandi si faccia riferimento all’**Appendice A**.

3.1.1 Elenco del contenuto di una directory

`ls`	elenca il contenuto della directory corrente
`ls -l` *`data`*	lista il contenuta della directory “data” fornendo maggiori dettagli
`ls -a`	mostra anche i file cosiddetti “nascosti” che iniziano con il “.”

3.1.2 Cambiamento di directory

cd */user/silv*	spostati nella directory “silv”

cd.. posizionati nell directory immediatamente superiore

cd vai nella directory indicata dalla variabile HOME ovvero la directory di login

3.1.3 Mostra la directory corrente

pwd Mostra la directory (di lavoro) corrente

3.1.4 Crea una directory

mkdir *lettere* crea la subdirectory “lettere” nella directory corrente

3.1.5 Cancella una directory

rmdir *personal* cancella la directory “personal”. Tale directory non deve contenere file. Eventualmente i file devono essere preventivamente cancellati con il comando rm personal/*

rm -r *trash* cancella recursivamente la directory “trash” con tutto il suo contenuto

3.1.6 Rinomina/sposta una directory o un file

mv *file1 file2* cambia il nome ad una directory ovvero sposta un file da un posto ad un altro

3.1.7 Visualizza il contenuto di un file

cat *old_data* lista il contenuto del file “old_data” sul video

3.1.8 Copia un file

cp *data1 data2* copia il contenuto del file “data1” sul file “data2”

3.1.9 Cancella un file

rm *data* cancella il file di nome “data”

rm -i *ant* chiede conferma per la cancellazione del file “ant”

ATTENZIONE questo comando è MOLTO PERICOLOSO . il comando “`rm *`” cancella tutto quello che avete nella directory corrente. Usate sempre l’opzione “`-i`” in modo che vi venga chiesta conferma.

3.2 Controlli di accesso

L’accesso ai file e directory in Unix vengono controllati dai privilegi o permessi associati ai file e alle directory. Questi permessi sono:

Privilegi	File	Directory
r read	dà accesso al contenuto del file	si possono vedere i file contenuti nella directory
w write	consente la modifica del contenuto del file o la sua cancellazione	si possono creare e cancellare file nella directory
x execute	il file può essere eseguito come un comand Unix	è possibile usare il comando cd per "andare" in quella directory, fare ricerche e copiare file.

Ad ogni file viene attribuito, inoltre, un proprietario (**user**) il quale afferisce ad un gruppo (**group**). Per visualizzare questi attributi potete digitare il comando

```
ls -l
-rw-r-----  1  root  bin   1003  Nov 12 18:30   file.1
-rwxr-x--x  1  root  bin   3790  Nov 12 18:30   program.2
drwxr--r--  1  root  bin   512   Nov 12 18:30   text
```

Come si può notare ad ogni file sono associate delle informazioni, il primo carattere all'estrema sinistra identifica il tipo di file ("d"=directory, "l"=symbolic link, "-"= file) i rimanenti nove caratteri formano tre insiemi di tre caratteri corrispondenti ai permessi relativi al proprietario (**u**ser), al gruppo (**g**roup) ed a tutti gli altri (**o**thers). Il comando per alterare i permessi ai file e directory è **chmod**, mentre per cambiare l'owner ed il gruppo bisogna ricorrere rispettivamente ai comandi **chown** e **chgrp**. Per la loro sintassi consultare l'**Appendice A**.
Attenzione è possibile con il comando chmod escludervi da un ulteriore accesso ad un vostro file.

3.3 Backup

Per procedure di Backup si intendono tutte quelle operazioni che hanno lo scopo di salvaguardare l'integrità dei dati presenti nei dischi magnetici di un dato computer. Lo Unix ha dei programmi (tar, cpio, dump) che realizzano tale fine ma quello di gran lunga più utilizzato è l'utility "tar". Vi sono poi varie utilities con lo stesso scopo ma non fanno parte di Unix.

3.3.1 tar

Per le operazioni di salvataggio dati su nastro Unix mette a disposizione l'utility tar che ha funzione di archiviare directory e file. Ecco alcuni esempi sul suo uso in diverse condizioni.

`tar cv`
scrive sull'unità a nastro di default il contenuto della directory corrente comprese le eventuali sub-directories (device di default /dev/rmt0h)

`tar cvf` ***`nomefile.tar`***
scrive sul file "nomefile.tar" il contenuto della directory corrente comprese le eventuali sub-directories

`tar cvf /dev/rmt1h`

scrive sull'unità a nastro /dev/rmt1h *il contenuto della directory corrente comprese le eventuali sub-directories*

`tar cvf /dev/nrmt1h`
accoda sull'unità a nastro /dev/nrmt1h *il contenuto della directory corrente comprese le eventuali sub-directories.*

ATTENZIONE la "n" in "nrmt1h" in quest'ultimo comando ha il significato di "NO REWIND"; questo permette di accodare su uno stesso nastro più archivi di tipo tar. Alla fine delle operazioni per riavvolgere il nastro dare il comando:

`mt -f /dev/nrmt1h rewind`

`tar xv`
ricrea nella directory corrente il contenuto del nastro nell'unità di default

`tar xvf /tmp/`*`nomefile.tar`*
ricrea nella directory corrente il contenuto del file "nomefile.tar"

`tar xvf /dev/rmt1h`
ricrea nella directory corrente il contenuto del nastro nell'unità /dev/rmt1h

Nel caso che su uno stesso nastro siano state registrate più sessioni tar con il comando "`tar cvf /dev/nrmt1h`" per recuperarle bisogna dare il seguente comando più volte in successione:

`tar xvf /dev/nrmt1h`
ricrea nella directory corrente il contenuto del nastro nell'unità /dev/nrmt1h

Per un più completo elenco delle sue funzionalità consultate la documentazione in linea con "man tar".

3.3.2 vbackup

E' un programma di utilità a corredo Sistema Operativo **Digital Unix** con lo scopo di facilitare lo scambio di dati e programmi tra macchine con il SO OpenVMS. In pratica non è altro che la versione Unix del programma **Backup** di OpenVms, con esso è quindi possibile leggere e creare nastri e files nel formato backup. Per il suo uso consultate l'help.

Capitolo 4

4. Comunicazioni

4.1 Il protocollo TCP/IP

Il TCP/IP[13] è un insieme di protocolli originariamente progettati per il Dipartimento della Difesa Americano (DoD) allo scopo di creare una rete di computers (che fu chiamata ARPANET) che permettesse ai partners governativi la condivisione delle risorse informatiche, a quel tempo scarse e molto onerose dal punto di vista finanziario. Con l'introduzione del TCP/IP nel SO UNIX il suo utilizzo si è rapidamente diffuso, specialmente nelle LAN, divenendo uno standard "de-facto" ed oggigiorno non esiste computer che non sia in grado di utilizzare questo tipo di protocolli.
L'originaria ARPANET fu smantellata nel 1988 e rimpiazzata da NFSNET[14] che si è espansa fino a diventare il "backbone" (dorsale) di una confederazione di reti locali, regionali e nazionali basata sul TCP/IP, chiamata **Internet**.

4.2 Telnet

TELNET è uno dei servizi forniti da questa suite di protocolli ed è l'equivalente del "set host" decnet. Viene usato per aprire sessioni su altri computers che riconoscono tale protocollo. Esempio:

```
$ telnet vxcern.cern.ch.
```

oppure

```
$ telnet
$ telnet>open vxcern.cern.ch.
```

Gli indirizzi TCP/IP sono di tipo gerarchico e consistono in stringhe di caratteri divise da punti. Nel nostro caso "vxcern" rappresenta il nome del computer o *host remoto* (come si dice in gergo) mentre "cern.ch" è il *dominio*[15] al quale appartiene; in questo caso il CERN (cern) con sede in Svizzera (ch). Poiché ai computers interessano più numeri che caratteri "vxcern.cern.ch" viene tradotto per mezzo di un *Nameserver* (BIND) in una sequenza di numeri (del tipo 128.141.100.1) che è quella usata per attuare l'effettivo collegamento.

Con il tasto *Ctrl-]* (o ESC di solito) potete tornare in qualsiasi momento al prompt `telnet>` dopo il quale possono essere digitati alcuni comandi quali ad esempio:

quit termina il programma telnet

[13] **T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol

[14] **N**ational **S**cience **F**oundation **NET**work

[15] insieme di computer appartenenti ad una medesima organizzazione

open apre una sessione con un host remoto

close chiude la sessione con l'host remoto

help visualizza tutti i comandi disponibili

4.3 Ftp

Un altro dei protocolli del TCP/IP è l'FTP[16] che viene sfruttato dall'utility che prende il suo stesso nome. Questo programma è lo strumento primario utilizzato per la copiatura di files da un computer all'altro attraverso Internet. Il suo uso è molto semplice, ecco un esempio di una sessione ftp:

```
$ ftp calf30.ca.infn.it.
...messaggio di benvenuto…
ftp> user il_vostro_username
password>la_vostra_password   (invisibile)
…messaggio…
*dir                          (lista della directory)
*cd math                      (vai nella directory math)
*dir                          (lista della directory)
*binary                       (trasferimento binario)
*get nome_del_file            (copia il file)
*quit                         (chiude la connessione)
```

Al prompt `ftp>` possono essere digitati molti comandi alcuni di essi sono:

quit termina il programma ftp

cd cambia directory sull'host remoto

lcd cambia directory sull'host locale

mkdir crea una directory sull'host remoto

pwd visualizza il path corrente completo sull'host remoto

put copia un file dall'host locale a quello remoto

mput copia più file dall'host locale a quello remoto

get copia un file dall'host remoto a quello locale

mget copia più file all'host remoto a quello locale

binary trasferisce i dati senza operare nessuna conversione

ascii trasferisce i dati in modo ASCII cioè file di testo

hash attiva il tracing del trasferimento con la visualizzazione del carattere "#" per ogni 1024 Bytes.

help visualizza tutti i comandi disponibili

Molto usato nella comunità di Internet è l'**ANONYMOUS FTP**; una convenzione per rendere disponibili a tutti coloro che hanno accesso alla rete una grande mole di

[16] **F**ile **T**ransfer **P**rotocol

programmi e documenti di vario genere. Questo mutuo accordo stabilisce che lo *username* da specificare nel comando ftp è "**anonymous**" e la password *l'indirizzo di posta elettronica* dell'utente che apre la connessione. In questo momento sono diverse migliaia i siti che offrono questo tipo di servizio in tutto il mondo.

4.4 Altri Servizi

Esiste un tipo di comandi (rsh, rlogin, rcp) che riguardano la comunicazione tra computers ai quali è associato un meccanismo di sicurezza tramite il file **.rhosts**. In questo file devono essere inseriti i nomi delle macchine remote e quello dei relativi utenti che sono abilitati ad usare il vostro account e quindi il computer *senza* password. Per esempio se il file **`.rhosts`** presente nella directory di login dell'account **gianni** nell'host **`saturn.ca.infn.it`** contiene le seguenti righe

```
#file .rhosts
hpp.cern.ch  mario
vaxca2.unica.it lorenzo
```

allora l'utente "mario" dal computer "hpp.cern.ch" e l'utente "lorenzo" dal computer "vaxca2.unica.it" avranno accesso al computer **`saturn`** usando l'account gianni.

Avvertenza: Per motivi di sicurezza è bene dare al file **.rhosts** il privilegio di lettura per il solo proprietario (*chmod 600 .rhosts*) in modo da evitare possibili intrusioni.

4.4.1 rsh

Esegue un comando su un host remoto. Esempi:

`rsh xxx.ca.infn.it df`

esegue il comando "df" sull'host xxx.ca.infn.it

`rsh -l silv host_rem date`

esegue il comando "date" sull'host xxx.ca.infn.it usando l'utente silv

4.4.2 rlogin

Qusto comando è usato per entrare nel vostro account residente su un host remoto. Per esempio se siete attualmente collegati col computer host1 con lo username giorgio e volete consultare il vostro account (con lo stesso username) che si trova nell'host host2 potete digitare

`rlogin host2`

In questo caso il file /etc/hosts.equiv deve contenere l'indicazione del computer chiamante host1.
Se invece volete usare un'altro username userete

`rlogin -l vanessa host2`

E' ovvio che in questo caso dovrete fornire la password dell'account vanessa.

4.4.3 rcp

Il programma rcp (remote copy) copia un file od una directory da un host ad un altro appartenti alla stessa rete. Ecco degli esempi sul suo uso:

`rcp myfile silvio@arca.ca.infn.it:myfile`

Per copiare il file myfile dal computer nel quale vi trovate nella home directory dell'account silvio dell'host arca.ca.infn.it

rcp *myfile arca.ca.infn.it:myfile*

Per copiare il file myfile dal computer nel quale vi trovate nella home directory dell'account, il cui username è uguale a quello nel quale vi trovate, dell'host arca.ca.infn.it

rcp *-r mydir silvio@arca.ca.infn.it:*

Per copiare la dir mydir e tutte le sue sub-dir dal computer nel quale vi trovate nella home directory dell'account silvio dell'host arca.ca.infn.it

rcp *ted@hp9.cern.ch:myfile bert@arca.ca.infn.it:myfile*

Per copiare il file myfile dalla home dir dell'account ted dell'host hp9.cern.ch nella home directory dell'account bert dell'host arca.ca.infn.it

NOTA: **rcp** non funziona se nel file di inizializzazione della shell nella vostra home directory (per esempio .cshrc per la C-shell) è presente un comando di output (ad esempio **echo**).

4.5 Stampa

La stampa dei files in Unix viene gestita tramite i comandi **lpr** (o **lp**), **lpq** e **lprm**. Il primo mette in coda nello *spooler* di stampa i files; il secondo interroga lo spooler allo scopo di ottenere una lista dei jobs pendenti per quella particolare stampante; mentre il terzo viene usato per cancellare i jobs dalla coda. Nell'AlphaServer **AFCA1**, per esempio, per praticità sono stati definiti dei comandi brevi (per mezzo degli alias) per ogni tipo di stampante e tipo di stampa.

Comando	Equivalenza	Stampante	Tipo
lps4	lpspr -N2 -K2 file | lpr -Ptps20 -h	TurboPrinterserver 20	Auto, PS,2s,2p
lps2	lpspr -K2 file | lpr -Ptps20 -h	TurboPrinterserver 20	Auto, PS,2s,1p
lps1	lpspr -K1 file | lpr -Ptps20 -h	TurboPrinterserver 20	Auto, PS,1s,1p
lpst	lpspr -K1 -Dascii file | lpr -Ptps20 -h	TurboPrinterserver 20	TXT,2s,1p
lpslc	lpspr -K1 -Dascii file -O landscape | lpr -Ptps20 -h	TurboPrinterserver 20	TXT,2s,1p
plw	lpr -Plw	LaserWriter IIg	Auto,1s,1p
plw6	lpr -Plw6	LaserWriter 4/600	Auto,1s,1p
pt8	lpr -Ppt8	TQ$LTA8	Auto,1s,1p
ps0	lpr -Pps0	HLASER	Auto,1s,1p
hpps	lpr -Phpps	HPLJET 600dpi	PS,1s,1p
hpt	lpr -Phptxt	HPLJET 600dpi	TXT,1s,1p

***PS** = Postscript; **TXT** = file ASCII; **AUTO**= selezione automatica del tipo di file; **2S** = stampa su entrambe le facciate del foglio (solo Printserver 20); **1P/2P**= stampa una o due pagine per facciata; **LAND** = Landscape; **R** = riduzione del 30%.*

Per formattare l'output sulla stampante esiste il comando filtro standard **pr** per ulteriori informazioni fate riferimento all'help. Digital Unix, il SO Unix della DEC, possiede in più il comando **lpspr** (programma filtro da usare in congiuzione con lpr) ed il suo omologo **xlpsprint** in ambiente X-Window.

4.6 Mail

Insieme al SO Unix viene fornito un programma per la spediszione di messaggi verso utenti di altri hosts. Di solito il suo nome è **mail**. Di seguito si elencano alcuni rudimenti sul suo utilizzo.

mail indirizzo_di_destinazione ...testo... .	spedire un messaggio, l'indirizzo deve essere nella solita forma standard *utente@mmm.xxx.yyy.cc* Il punto, da solo all'inizio della riga, indica la fine del messaggio.
mail	leggere un messaggio
d	cancellare il messaggio corrente
s filename	scrivere il messaggio corrente in un file
q	uscire dal programma mail
mail dest < filename	spedire come messaggio il contenuto del "filename" all'indirizzo "dest"

Esiste anche il comando **mailx** (o **Mail**) combinazione delle utility di mail tra la versione Unix BSD e quella System V che aggiunge altre funzionalitá al comando **mail** tra le quali le *liste di distribuzione* (insiemi di indirizzi verso la quale spedire un determinato messaggio) definite nel file di inizializzazione del mail ".mailrc".

mailx indirizzo_di_destinazione ...testo... .	spedire un messaggio
mailx	attivare il programma
p numero_del_messaggio	visualizzare uno dei messaggi dell'elenco
d	cancellare il messaggio corrente
R	rispondere ad un messaggio
s filename	scrivere il messaggio corrente in un file
q	uscire dal programma mail
mailx -s prova dest < filename	spedire come messaggio il contenuto del "filename" all'indirizzo "dest", "prova" é l'oggetto del messaggio.
mailx -f	leggi messaggi precedenti presenti nel folder di default
mailx managers < riunione.dat	Spedisce alla lista "managers" il file "riunione.dat". Nel file ".mailrc" sará definita un alias per managers nel modo seguente: alias managers gianni, silv@xxx.yyy.it, roberto

E' anche possibile il *forwarding* cioè la ritrasmissione automatica dei mail in arrivo verso altra destinazione. Questo scopo è raggiunto con l'ausilio del file **.forward** che deve risedere nella directory di login dell'utente. Tale file contiene uno o più indirizzi (nella forma standard per internet) verso i quali ritrasmettere i messaggi in arrivo.

4.7 Pine

Pine è un *mail user agent* di pubblico dominio molto diffuso e semplice da usare ed è in grado di gestire una grande quantità di messaggi. Pine supporta MIME[17] ed è quindi in grado di ricevere e spedire messaggi multimediali. Cioè oltre che testi si possono inviare anche immagini, suoni e programmi binari.

[17] **M**ultipart **I**nternet **M**ail **E**xtensions

Capitolo 5

5. SOFTWARE

5.1 Compiling and linking

Compilare e linkare programmi (Fortran o C) in Unix è un po' differente da quello in OpenVms. Sono possibili tre modi:

- Compilare e "linkare" con un unico comando. Questo è il metodo più usato.
- Compilare e linkare con comandi separati. Metodo poco usato dato che il linker Unix (**ld**) viene chiamato per default dal compilatore.
- Compilare e Linkare con l'utility **make**. Questo metodo viene di solito usato per compilazioni lunghe e complesse. Questo sistema esiste anche in OpenVms che sfrutta l'utility **mms**.

Esempi di compilazioni:

```
f77  hello.f
cc  hello.c
```

entrambi i comandi producono come output un file eseguibile `a.out` come default. Se si desidera un nome file di output diverso dare

```
f77 -o hello hello.f
cc -o hello1 hello.c
```

Se il vostro programma sorgente è diviso in varie routine il comando è

```
f77 -o analyze main.f routine1.f  routine2.f … routinen.f
```

nel caso volete creare gli object files a partire dai sorgenti, non facendo eseguire la fase di link, userete l'opzione "-c"

```
f77 -c routine.f
```

questo comando esegue solo la compilazione del programma `routine.f` e crea il file object `routine.o`, che può successivamente essere inserito in comando di compilazione o di link

```
f77 -o funz main.f routine.o
```

5.2 Librerie

Le librerie di programmi sono insiemi di file raggruppati. In Unix il programma di manipolazione delle librerie è `ar`. Di solito il nome delle librerie inizia con "**lib**" e termina con l'estensione "**.a**". l'estensione può anche essere di tipo "**.sl**" che sta per

shared library.
Nello specificare librerie in un comando di compilazione e link si possono usare due tecniche. Esempio di compilazione con librerie CERN:

```
f77 main_prog.f  sub.f /cern/pro/lib/libpacklib.a
```

oppure con l'equivalente

```
f77 main_prog.f sub.f -L/cern/pro/lib -lpacklib
```

l'opzione "-L" individua la directory dove cercare le librerie e l'opzione "-l" specifica la particolare libreria **lib**packlib.**a** (o **lib**xxx.**sl**) da usare.

5.2.1 Il comando cernlib

Quest comando disponibile in tutti quei sistemi nel quale sono installate le librerie del CERN ha lo scopo di facilitarne l'accesso. In pratica esso restituisce il pathname completo delle librerie che sono state specificate come suoi argomenti. Esempio:

```
cernlib mathlib packlib
/cern/pro/bin/libmath.a /cern/pro/bin/libpacklib.a
```

In questo modo si può costruire un comando di compilazione e link usando questo comando

```
f77 -o testmain prog.f 'cernlib mathlib packlib'
```

Si raccomanda nello scrivere programmi di fare compilazioni con le opzioni di ottimizzazione in stato di "off" allo scopo poi di poter usare il *debugger* (**dbx**).
Per rendere possibile il *debugging* è necessario compilare i programmi con l'opzione "-g".

Capitolo 6

6. Come fare per ...

Questa sezione cerca di rispondere a quelle domande che un utente potrebbe porre su argomenti specifici e che per motivi di spazio non sono stati approfonditi nei capitoli precedenti.

6.1 Cancellare un file il cui nome inizia con "-" ?

Se un file inizia con il simbolo "-" come ad esempio "-filedat.txt" non è possibile cancellarlo con "rm -filedat.txt" perchè la stringa "filedat.txt" verrebbe interpretata come un opzione del comando "rm". Per evitare questo il modo più semplice è

```
rm ./-filedat.txt
```

o in alternativa, ma non è detto che funzioni con tutte le versioni di Unix, i comandi

```
rm - -filedat.txt
```

o

```
rm -- -filedat.txt
```

6.2 Cancellare un file con caratteri "strani" nel nome ?

Succede di imbattersi (purtroppo) in nomi di file con caratteri di controllo, spazi e slashes (/). In questi casi il modo più semplice è quello di cercare una sequenza che selezioni univocamente il file che contiene il carattere "strano" come ad esempio:

```
rm -i file*name*.xxx*.yyy
```

6.3 Inserire il pathname corrente nel prompt ?

csh Inserire nel file .cshrc le seguenti righe:

```
alias setprompt ´set prompt="${cwd}% "´
setprompt
alias cd ´chdir \!* && setprompt´
```

sh creare un nuovo comando per cd per esempio xcd definito come

```
xcd() { cd $* ; PS1="'pwd' $ "; }
```

da inserire nel file .profile

ksh mettete nel vostro file .profile la definizione
PS1='$PWD $ '
se volete solo l'ultimo componente del pathname dovete inserire
PS1='${PWD##*/ }$ '

tcsh set prompt='%/' Per tutto il pathname
set prompt='%c' Per l'ultima parte del pathname

bash PS1='\w $ ' Per tutto il pathname
PS1='\W $ ' Per l'ultima parte del pathname

6.4 Rinominare files "*.xyz" in "*.vwk" ?

Un comando del tipo "`mv *.dat1 *.dat2`" (che si suppone possa rinominare tutti i file con estensione dat1 con l'estensione dat2) fallisce in Unix poichè (a differenza di quello che succede in OpenVms) gli argomenti vengono espansi PRIMA che vengano passati al programma "`mv`". Di conseguenza se avete bisogno di rinominare un gran numero di files dovete ricorrere ad un vero e proprio programma per una delle shell Unix (csh, ksh, bash, tcsh, ecc.).
Primo esempio: come simulare `rename *.xyz *.vwk`

C shell
```
foreach f ( *.xyz )
      mv $f $f:r.vwk
end
```
Korn shell
```
for f in *.xyz; do
      mv $f ${f%xyz}vwk
done
```

Secondo esempio: convertire nomi files da maiuscolo a minuscolo

C shell
```
foreach f ( * )
      mv $f ‘echo $f | tr ’[A-Z]’ ’[a-z]’‘
end
```
Bourne Shell
```
for in *; do
      mv $f ‘echo $f | tr ’[A-Z]’ ’[a-z]’‘
done
```
Korn shell
```
typeset -l l
for f in *; do
      l=“$f”
      mv $f $l
done
```

6.5 Trovare la data di creazione di un file ?

In Unix questo è impossibile, infatti, la data di creazione di un file non viene registrata in nessun posto. Le uniche date che possono essere rese visibili sono: la data dell'ultima modifica (comando **`ls -l`**), la data dell'ultimo accesso (comando **`ls -lu`**) e la data di variazione dell'I-node (comando **`ls -lc`**).

6.6 Recuperare un file cancellato con "rm" ?

Fate molta attenzione, questo comando è potenzialmente pericoloso poichè basta digitare `rm * .dat` con uno spazio tra l'asterisco ed il punto e avrete cancellato TUTTA LA VOSTRA DIRECTORY. Non esistono metodi pratici per recuperare i files cancellati. E' quindi buona regola usare l'opzione "`-i`" del comando "`rm`", in tal modo vi viene chiesto conferma prima di ogni operazione di cancellazione.

6.7 Eseguire programmi con input da uno script o in background ?

Programmi interattivi del tipo telnet, ftp, passwd, ecc. si aspettano un input da terminale e le varie Shells non foniscono nessun mezzo per emulare questo tipo di input. Per simulare l'input da terminale si deve ricorrere ad un altro programma. Uno

di questi programmi è “expect” che è possibile trovare via rete (per esempio a ftp.cme.nist.gov).

6.8 Riportare la data in un nome di file ?

Per fare questo bisogna rivedere il comando date il quale permette di modificare o visualizzare la data corrente. Questo programma utilizza una stringa di formattazione che serve per variare la presentazione della data medesima; per ulteriori informazioni si consultino le man pages. Potete così, per esempio, creare un file con la data separata da un punto inserendo la seguente riga in uno script.

```
FILENAME=report.`date ´+%d%m%y´`
```

6.9 Come scambiare dati con nastro tra UNIX e OpenVMS ?

Per scambiare dati tra Unix e OpenVms si possono usare le utility **tar**, **vbackup** (vedi sezione 3.3.2) o **ltf**.

6.9.1 tar

Questa utility nel SO OpenVMS non esiste ma è disponibile come prodotto freeware (di pubblico dominio).
Se si vogliono salvare i file della directory “data1” per leggerli successivamente con un computer di tipo Unix procedere nl modo seguente:

```
$ mount /fore/reco=512/block=10240 Mkx00:
$ tar cvf Mkx00: [.data1]
$ dism Mkx00:
```

Per estrarre da un nastro scritto da una macchina Unix nella directory *target_directory* con l’utility tar procedere invece come segue:

```
$ set def target_directory
$ mount /fore/reco=512/block=10240 Mkx00:
$ tar xvf Mkx00: [.data1]
$ dism Mkx00:
```

Potete anche selezionare un singolo file o gruppo di file

```
$ set def target_directory
$ mount /fore/reco=512/block=10240 Mkx00:
$ tar xvf Mkx00: [.data1]*.dat
$ dism Mkx00:
```

6.9.2 ltf

Un altro metodo abbastanza semplice è quello di utlizzare l’utility **ltf** (labelled tape facility) la quale fornisce un metodo standard per esportare files verso Sistemi Operativi diversi da Unix. Per saperne di più consultate l’help con “man ltf”.
Per esempio se volete esportare dei files verso OpenVms digitate il comando

```
ltf -c /dev/rmt0h file1 file2 *.for
```

il quale trasferirà i files selezionati sul nastro.
Successivamente con un sistema VMS “montate” il nastro con

```
mount /over=id  mkxnnn:
dir
```

ed ecco i files pronti all'uso.
E'naturalmente valido anche il viceversa, per leggere nastri con macchine UNIX scritti da sistemi VMS non dovete fare altro che usare il comando

```
ltf -x  vmsfile1 vmsfile2 *.for
```

il quale copierà nel vostro sistema UNIX (dall'unità a nastro di default /dev/rmt0h) il files selezionati.
Nel caso volete sapere il contenuto di nastro (dall'unità di default /dev/rmt0h) il comando da digitare è

```
ltf -t
```

6.10 Trovare il process ID di in programma ?

La maniera più semplice di far questo è quella di fare una ricerca nella lista dei processi di vostra proprietà con il comando

ps ux | grep *nomeprogramma*

dove name è il nome del processo di cui volete trovare il *process id*.

Appendice A

UNIX per ESEMPI

COMANDO	DESCRIZIONE
	Files
ls	Elenca i files della directory corrente
ls -l	Elenca i files della directory corrente con le protezioni, il proprietario, la data e le dimensioni
ls -a	Elenca tutti i files della directory corrente compresi quelli nascosti. (Cioè i files che iniziano con il ".")
ls *.f	Elenca tutti i files con estensione ".f"
ls xyz?.dat	Elenca tutti i files con qualsiasi carattere al posto di "?"
ls ~	Lista il contenuto della home directory
ls [a-h]*	Lista tutti i files che iniziano con le lettere dalla "a" alla "h"
ls /usr/bin	Lista il contenuto della directory "/usr/bin"
ls -R	Lista il contenuto della directory corrente e di tutte le sue (eventuali) sub-directories.
cat *nomefile*	Lista il contenuto del file "*nomefile*"
cat -n *nomefile*	Lista il contenuto del file "*nomefile*" con le linee numerate
more *nomefile*	Lista il contenuto del file "*nomefile*" una pagina alla volta. Il tasto SPAZIO fa avanzare di una pagina, il tasto "b" fa tornare indietro di una pagina, il tasto RETURN fa avanzare di una linea, la sequenza "/radio" cerca la stringa "radio" e "q" termina il programma.
head *nomefile*	Lista le prime linee di "*nomefile*"
tail *nomefile*	Lista le ultime linee di "*nomefile*"
tail -40 *nomefile*	Lista le ultime 40 linee di "*nomefile*"
less	programma simile a more
	Directory
pwd	Lista il nome della directory corrente
mkdir *source*	Crea la directory "*source*" nella directory corrente
cd *source*	Spostati nella directory "*source*"

cd *source*/*for*	Spostati nella directory "*for*" contenuta nella directory "*source*"
cd	Vai nella home directory (o di login)
cd ..	Spostati nella directory immediatamente superiore
rmdir *source*	Cancella la directory "*source*" (che deve essere vuota)
	Operazioni con File e Directory
cp *file1 file2*	copia il "*file1*" nel file "*file2*" (se *file2* non esiste viene creato)
cp /*usr*/*bin*/*ktx* .	copia il file "*ktx*" della directory "/*usr*/*bin*" nella directory corrente. (Rappresentata da ".")
cp *file1* ~	copia *file1* nella directory di login (o home directory). Il simbolo "~" rappresenta la home directory dell'utente.
cp *file1 source/for/data*	copia *file1* nella directory "*data*"
cp -r *data /disk1*	copia la directory data con tutto il suo contenuto nella directory *disk1*. Alla fine avremo una directory */disk1/data*.
cat *file1 file2 > file3*	concatena *file1* e *file2* nel file "*file3*"
rm *file1*	cancella *file1*
rm *.*o*	cancella tutti i files con estensione "*.o*"
rm -i *.*f*	cancella tutti i files con estensione "*.f*" chiedendo per ogni file la conferma
rm -r *nomedirectory*	cancella la directory "*nomedirectory*" e tutto quello che essa contiene (directories e files)
mv *file1 file2*	rename *file1* come *file2*
mv *file1 source/text*	sposta il *file1* nella directory "*text*"
ln *myprog prog1*	Un nuovo link "*prog1*" (cioè un nuovo nome) viene creato per il file "*myprog*"
ln *file1 dir*	crea un link chiamato "*file1*" nella directory "*dir*" verso il file "*file1*". Questo tipo di link è chiamato *hard* perchè è possibile solo per files e nell'ambito di uno stesso *file system*
ln -s *file1 dir*	crea un link chiamato "*file1*" nella directory "*dir*" verso il file "*file1*". Questo tipo di link è chiamato *soft* perchè, al contrario del precedente, non è legato all'ambito di un solo *file system* e puó creare anche link tra directory
ln -s *dir1 dir2*	crea il link "dir2" che punta verso la directory "dir1"
file *nome*	fornisce informazioni sul file "nome"
	Editors & Terminali
vi *nomefile*	Edit *nomefile* con l'editor "vi". "i"= inserisce testo, "Esc"= ritorna in modo comando, "x"=cancella carattere,

	":q!"= esce senza registrare i cambiamenti, ":wq"=esce salvando il file.
vi -r *nomefile*	recupera gli ultimi cambiamenti effettuati su "*nomefile*" dopo un interruzione anormale.
emacs *nomefile*	richiama l'editor emacs
edt *nomefile*	richiama la versione Unix (Digital Unix) dell'editor OpenVms EDT
tset *vt100*	predispone l'emulazione VT100
tset -e^H	associa il carattere DELETE al tasto "BACKSPACE"
tset -e^\?	associa il carattere DELETE al tasto "DELETE"
Comunicazioni	
telnet *vaxca* telnet *vxcern.cern.ch*	apre una sessione su "*vaxca*" o "*vxcern.cern.ch*" con il protocollo tcp/ip.
rlogin *afca1* rlogin *afca1.ca.infn.it*	apre una sessione su "*afca1*" con lo username corrente. Se nella directory di login di afca1 il file ".rhosts" contiene la definizione dell'host dal quale avete dato il comando non viene chiesta nessuna password.
rlogin *afca1* -l *dart*	apre una sessione con "*afca1*" usando lo username "*dart*"; viene richiesta la password
ftp *afca1.ca.infn.it*	attiva l'utility "ftp" per il trasferimento di files dall'host (o verso l'host) "*afca1*". I comandi fondamentali sono: "dir", "cd", "put", "get", "binary", "mget", "mput".
ftp -i *afca1.ca.infn.it*	disabilita la conferma per trasferimenti multipli con "mget" o "mput".
rcp *afca1:source/data.f* .	copia dalla directory "*source*" (contenuta nella mia directory di login) nell'host "*afca1*" il file "*data.f*" nella directory corrente (.)
rcp *data.f afca1:/source*	lo stesso di prima ma nella direzione opposta
rsh *afca1 ls*	esegue il comando "*ls*" nell'host "*afca1*"
finger *ute@xx.ca.infn.it*	permette di sapere se l'utente "ute" è attualmente collegato all'host remoto "xxx.ca.infn.it" oppure a quando risale la data dell'ultimo login.
logout	chiude la sessione
exit	chiude la sessione esce dalla shell
Sicurezza	
passwd	cambia la password
chmod *u=x nomefile*	rende "*nomefile*" eseguibile dal solo proprietario
chmod *a+r nomefile*	aggiunge il permesso di eseguire "*nomefile*" da parte di chiunque

chmod *g-r nomefile*	toglie l'accesso in lettura a "*nomefile*" da parte di tutti i componenti del gruppo al quale appartiene
chmod *o+w nomefile*	aggiunge il permesso di eseguire "*nomefile*" da tutti
chmod *g-x nomefile*	toglie il permesso di esecuzione a "*nomefile*" da parte di tutti i componenti del gruppo
chown *owner file*	cambia la proprietà del "*file*"
chown *owner:group file*	cambia cotemporaneamente la proprietà ed il gruppo di appartenenza del "*file*"
chgrp *group file*	cambia l'appartenza al "*group*" del "*file*"
Pipes e re-indirizzamento dell'output	
ls -l | more	lista i files della directory corrente una pagina alla volta. Il simbolo "|" rappresenta una pipe.
ls -l | cat -n > *ls.out*	lista i files, numera ogni linea e scrive l'output nel file "*ls.out*"
prog < *pro.dat* >*pro.out*	esegue il programma "prog" che prende i dati di input dal file "*pro.dat*" e scrive l'output nel file "*pro.out*"
prog >> *prog.out*	accoda l'output de programma "prog" al file "*prog.out*"
Gestione stampe	
lpr *aaa.f*	stampa il file "*aaa.f*" sulla stampante di default
lpr -P*argo stat.c*	stampa il file "*stat.c*" sulla stampante "*argo*". L'opzione "-P" individua la stampante ed è obbligatorio se non ne è stata definita una di default
prog | lpr	stampa l'output del programma "prog"
lpq	lista il contenuto della coda per la stampante di default.
lpq -Pargo	lista il contenuto della coda per la stampante "argo"
lprm *number*	cancella il job numero *number* dalla coda di stampa. Il *number* viene desunto dall'output del comando lpq.
pr [*opzioni] filename*	programma filtro con numerose opzioni per formattare testi e stamparli
Ricerche di stringhe e files	
grep OPEN *.f	cerca tutte le righe con la parola "OPEN" (o delle parole che soddisfano una data *regular espression*) in tutti i files con estensione ".f" nella directory corrente (i caratteri maiuscolo e minuscolo sono considerati diversi)
egrep '(^[a-zA-Z]|xhc)' \ ricambi	la funzione di egrep (*expression grep*) è simile a quella di grep ma in più prevede la possibilità di ricercare righe che soddisfano DUE *regular expression.* Nell'esempio si cercano le righe nel file "ricambi" che iniziano con una qualsiasi coppia di caratteri o con "xhc"

fgrep *-i -f ttt file2*	la funzione fgrep (*file based grep*) è quella di cercare diverse (più di due) configurazioni nel file "file2" con l'ausilio del file "ttt"
find / -name "*.c" -print	cerca i files che hanno la parte finale che finisce per ".c" sulla root directory e su tutte le sue subdirectories
find . -name "*.f" -print	cerca tutti i files che finiscono per ".f" nella directory corrente e tutte le sue subdirectories
find . -mtime -2 print	cerca tutti i files che sono stati modificati negli ultimi 2 giorni
find . -name "*.f" -exec grep VAR {}\;	cerca in tutti i file *.f nella directory corrente ed in quelle sottostanti la stringa "VAR"
	Backup
tar cv	scrive sull'unità a nastro di default il contenuto della directory corrente comprese le eventuali sub-directories
tar xv	legge dall'unità a nastro di default il contenuto del nastro e lo scrive nella directory corrente
tar xvf /data/arch1.tar	legge il contenuto del file "/data/arch1.tar" il viene "spacchettato" nella directory corrente
tar cvf /dev/nrmt1h	scrive sull'unità a nastro /dev/rmt1h il contenuto della directory corrente comprese le eventuali sub-directories
tar tv	lista sul video il contenuto del nastro nell'unità di default
tar tvf /dev/rmt0h	lista sul video il contenuto del nastro nell'unità "/dev/rmt0h"
tar cvf arch.tar	scrive sul file "arch.tar" il contenuto della directory corrente comprese le eventuali sub-directories
ltf	programma per lo scambio di dati su nastro tra SO e piattaforme hardware differenti (vedi sezione 6.9)
mt rewind	riavvolgi il nastro presente nell'unità a nastro di default (nrmt0h)
mt fsf *n*	salta al file *n +1*
mt -f /dev/rmt1h rewind	riavvolgi il nastro dell'unità a nastro nrmt1h
	Programmi Filtro
spell *filename*	esegue un check dello spelling del testo nel file "filename"
spell -b *filename*	come sopra ma usando le spelling Britannico
nl *nome1*	numera le righe del file "nome1"
wc *nome2*	conta le righe, le parole e i caratteri del file "nome2"
sort *test1*	ordina le righe del file "test1" in ordine alfabetico
sed -e *'s/toro/cane/g' test*	sostituisce tutte le parole "toro" con "cane" nel file "test", il risultato viene inviato nello standard output

Informazioni varie sul sistema	
df	lista le informazioni sull'uso delo spazio nei dischi
du	lista le dimensioni (in Kilobytes) di tutte le directory presenti nel pathname corrente ed in quelle sottostanti
who	elenca gli utenti collegati al sistema
whoami	visualizza logon ID
Sviluppo programmi	
f77 -o *art prog.f*	compila e linka il file "prog.f" creando l'eseguibile "art"
f77 -c *prog.f*	compila "prog.f" creando il file oggetto "prog.o"
f77 -o *main prog1.f*\ *project/lib/libpart1.a*	compila e link il programma "prog1", creando l'eseguibile "main" usando la libreria "lib1.a"
f77 -o *main prog1.f* \ -L*/project/lib* -l*part1*.a	Un'altra maniera per ottenere lo stesso effetto
f77 -g -c main.f f77 -g -c prog1.f f77 -o main main.o \ prog1.o	compila i programmi con l'opzione "-g" per usare il debugger. Produce l'eseguibile "main"
dbx main	Se l'esecuzione di main produce un errore si può usare il debugger "dbx" per cercare di individuarlo
Controllo processi	
prog **&**	esegue il programma "prog" in background; viene generata una riga con un numero (process identification number,Pid) che identifica univocamente il processo
<Ctrl-c>	cancella il processo/programma corrente
<Ctrl-z>	stop del processo/programma corrente; successivamnete può essere posto in background con **bg** o riportato in fore-ground con **fg**
ps	lista tutti i processi attivi con il vostro username. La lista contiene il Pid assegnato ai processi.
ps -xu o **ps xu**	come quello di prima ma più dettagliato
kill *Pid*	cancella il processo identificato dal *Pid,* questo è il modo più "pulito" di cancellare un processo
kill -1 *Pid*	Se il precedente non ha effetto si può provare ad inviare al processo il segnale (-1) che simula il logout. Il sistema tenterà di "uccidere" gli eventuali *child processes*
kill -9 *Pid*	come ultima risorsa si usi il segnale (-9). E' una procedura drastica che può lasciare *child processes* attivi
Help	
man tar	elenca tutte le informazioni sull'uso del programma "tar"

man -f tar	priduce una riga di informazioni sul comando “tar”
whatis ls	informa il tipo di azione eseguita dal comando “ls”
whereis mt	fornisce il pathname del comando “mt”
apropos *keyword*	lista tutti i comandi che sono in relazione con la *keyword*
man -k *keyword*	sinonimo di **apropos**

Appendice B

Corrispondenza tra comandi OpenVMS e Unix

Segue una tabella di corrispondenza tra comandi Unix e VMS. Naturalmente tale relazione non è perfetta data la diversa natura dei due SO, ma può essere utile per farsi un'idea del diverso approccio che questi hanno nei riguardi dell'utente.

OpenVMS	UNIX	Descrizione
Dir	ls	Elenca files
Dir /full	ls -l	Elenca files con l'aggiunta di altre informazioni
Dir /modifi/before (/since)	find	cerca e seleziona dei file in base alla data o ad altri parametri
Type	cat	lista file
Type /page	more	lista file una pagina alla volta
Type /tail	tail	lista le ultime righe di un file
Copy	cp	copia un file
Search	grep, fgrep	cerca una stringa in uno o più files
Diff	diff, cmp	confronta due files
Rename Backup/delete	mv	rinomina un file o sposta un file da un posto ad un altro
Delete	rm, rmdir	cancella un file o una directory
Set file /prot	chmod	cambia le protezioni di un file
Set file /own	chown	cambia la proprietà di un file o di una directory
Create /dir	mkdir	crea una directory
Set default	cd	cambia la directory di lavoro corrente
Show Default	pwd	visualizza la directory corrente
Help	man, apropos	richiama l'utility di help
Show time	date	visualizza la data e l'ora
Sho dev d	df	visualizza informazioni sull'uso dello spazio disco
dir /siz [...]	du	lista le informazioni sulle dimensioni dei files directory per directory a partire da quella corrente
Stop /id	kill	cancella un processo

Link	ld	crea un file eseguibile da moduli oggetto
Print	lpr	stampa un file
Show queue	lpq	visualizza le code di stampa
Show entry	lpq	visualizza le code di stampa
Delete/entry (/queue)	lprm	cancella jobs dalla coda di stampa
Set password	passwd	cambia la password
Show time	date	visualizza data ed ora
Show users	who, finger	elenca gli utenti collegati
Show process	ps	visualizza informazioni sui processi
Set terminal	stty	cambia i parametri del terminale
Phone	talk	chiama e conversa con un altro utente
Backup	tar	backup dei dati da disco a nastro (o altro disco)
Dump	od	lista file in vari formati (esadecimale, ottale, ecc)
Library	ar	crea e gestisce delle librerie
Recall/all	history	lista i comandi precedenti
Write sys$output	echo	visualizza il valore degli argomenti che seguono

Bibliografia

UNIX for VMS Users, Philip Bourne, Digital Press, 1990.

Teach Yourself UNIX in a week, Dave Taylor, Sams Publishing, 1994

Unix shell Programming, Stephen G. Kochan, Patrick H. Wood, Hayden Books, 1992.

Unix for the Impatient, Paul W. Abrahams, Bruce A. Laason, Addison & Wesley Publishing Company, 1992.

Unix, Quick!, Andrew Feibus, Professional Press Books, 1962.

A Practical Guide to the UNIX system, Mark G. Sobell,

Practical UNIX Security, S. Garfinkel & G. Spafford, O'Reilly, 1990

UNIX for Programmers, D. Farkas, Wiley, 1987.

Sed & Awk, Dale Dougherty, O'Reilly & Associates, Inc., 1992.

Learning Perl, Randal L. Schwartz, O'Reilly & Associates, Inc., 1994.

Efficient Fortran Programming, A. Kruger, Wiley & Sons, 1990.

Effective Fortran 77, M. Metcalf, Oxford University Press, 1985.

The Design and Implementation of Programs in Fortran 77, H. Lee, P. Munsell, Prentice Hall, 1990.

Fortran 90 Explained, M. Metcalf, J. Reid, Oxford University Press, 1990.

C For Fortran Programmers, T. D. Brown, Prentice Hall, 1990.

Linguaggio C, B. W. Kernighan, D. M. Ritchie, Jackson, 1989.

Learning GNU Emacs, Debra Cameron and Bill Rosenblatt, O'Reilly, 1992