Introduzione a Unix

Frank G. Fiamingo

Linda DeBula

Linda Condron

Introduzione a Unix

Frank G. Fiamingo
Linda DeBula
Linda Condron
Revisione: 43184
Settembre 1998 di .
Diritto d'autore © 1996, 1997, 1998 University Technology Services, The Ohio State University.

Sommario

Traduzione a cura di Marco Trentini <mark@remotelab.org >.

Questo documento introduce il vasto mondo dei sistemi operativi Unix e lo fa in stile manualistico. Unix non è un sistema operativo unico, come molti potrebbero pensare, bensì è una grande famiglia di sistemi operativi (come ad esempio FreeBSD, NetBSD, OpenBSD, BSD, AIX, System V, SunOS, varie distribuzioni di Linux, ecc.), aventi comandi e caratteristiche simili e disponibili per molte architetture hardware (come ad esempio x86, alpha, ecc.).

Copyright e URL rivisti nel Settembre 1998.

© 1996-1998 University Technology Services, The Ohio State University, Baker Systems Engineering Building, 1971 Neil Avenue, Columbus, OH 43210.

Tutti i diritti riservati. La ridistribuzione e l'uso, con o senza modifiche, sono permesse purchè siano rispettate le seguenti condizioni:

1. La ridistribuzione deve contenere la precedente nota di copyright, questa lista di condizioni, e il seguente disclaimer.

2. Né il nome dell'Università né i nomi dei suoi contributori possono essere usati per appoggiare o promuovere prodotti o servizi derivanti da questo documento senza un precedente specifico permesso scritto.

QUESTA PUBBLICAZIONE VIENE FORNITA «COSÌ COM'È» SENZA NESSUN TIPO DI GARANZIA. QUESTA PUBBLICAZIONE PUÒ CONTENERE IMPRECISIONI TECNICHE O ERRORI TIPOGRAFICI.

Unix è un marchio registrato della The Open Group, AT&T è un marchio registrato della American Telephone and Telegraph, Inc.

Indice

Prefazione vii
1. Storia di Unix 1
2. Struttura di Unix 3
    2.1. Il sistema operativo 3
    2.2. Il file system 4
    2.3. Directory, file e inode Unix 5
    2.4. Programmi Unix 5
3. Iniziamo 7
    3.1. Effettuare il login 7
    3.2. Struttura della linea di comando di Unix 9
    3.3. Tasti di controllo 10
    3.4. stty - controllo del terminale 11
    3.5. Ottenere aiuto 12
    3.6. Navigazione e controllo delle directory 14
    3.7. Comandi di gestione dei file 18
    3.8. Comandi di visualizzazione 25
4. Risorse di sistema e stampa 31
    4.1. Risorse di sistema 31
    4.2. Comandi di stampa 39
5. Shell 45
    5.1. Comandi built-in 45
    5.2. Variabili di ambiente 47
    5.3. La shell Bourne, sh 49
    5.4. La shell C, csh 50
    5.5. Controllo dei job 52
    5.6. History 53
    5.7. Cambiare la propria shell 55
6. Caratteristiche speciali di Unix 57
    6.1. Descrittori di file 57
    6.2. Redirezione di file 57
    6.3. Altri speciali simboli di comando 60
    6.4. Meta caratteri 61
7. Manipolazione del testo 63
    7.1. Sintassi delle espressioni regolari 63
    7.2. Comandi di manipolazione del testo 65
8. Altri comandi utili 77
    8.1. Lavorare con i file 77
    8.2. Archiviazione, compressione e conversione di file 96
    8.3. Connessioni remote 104
9. Programmazione di shell 111
    9.1. Script di shell 111
    9.2. Settare i valori dei parametri 111
    9.3. Quoting 112
    9.4. Variabili 113

9.5. Sostituzione di parametri 115
9.6. Here document 116
9.7. Input interattivo 117
9.8. Funzioni 118
9.9. Comandi di controllo 120
10. Editor 129
10.1. Configurare la propria sessione vi 129
10.2. Configurare la propria sessione emacs 130
10.3. Veloce guida per vi 132
10.4. Veloce guida per emacs 134
11. Riassunto dei comandi Unix 139
11.1. Comandi Unix 139
12. Una breve bibliografia Unix 151
Glossario 153

Lista delle tabelle

3.1. Comandi di navigazione e controllo delle directory 14
3.2. Comandi di navigazione e controllo delle directory Unix vs DOS 14
3.3. Comandi di gestione dei file 19
3.4. Comandi di gestione dei file Unix vs DOS 19
3.5. Comandi di visualizzazione 25
4.1. Comandi per le risorse di sistema 31
4.2. Comandi di stampa 39
5.1. Comandi di sostituzione di history per la shell C 54
6.1. Redirezione di file 58
7.1. Comandi di manipolazione del testo 65
8.1. Utilità file 77
8.2. Comandi di archiviazione, compressione e conversione di file 96
8.3. Comandi per connessioni remote 104
9.1. Variabili di shell 113
11.1. Comandi Unix 139
12.1. Una breve bibliografia Unix 151

Prefazione

Questo documento è rivolto in prevalenza a quelle persone che si avvicinano per la prima volta a Unix, ma non solo.

Prima di iniziare ad esporre la traduzione di questo documento volevo dire alcune cose. Alcune parole del testo originale in lingua inglese non sono state tradotte in italiano, vuoi perchè la rispettiva traduzione in italiano non è efficiente in termini di comprensibilità, vuoi perchè è di uso comune far riferimento a queste direttamente in lingua inglese (vedi login, quoting, built-in, here document, shell, background, pipe, script, ecc.). Inoltre mi è sembrato opportuno e logico modificare l'ultima sezione di questo documento, bibliografia di Unix (testi in lingua inglese), in una breve bibliografia italiana di Unix. Infine ho aggiunto un glossario dove poter trovare un elenco dei termini inglesi più caldi incontrati in questo documento e il loro significato corrispondente.

Non mi resta che augurarvi una buona lettura.

Convenzioni usate in questo libro

Per fornire un testo consistente e facile da leggere, sono state seguite numerose convenzioni in tutto il libro.

Convenzioni Tipografiche

Italico
: Un font *italico* è per i nomi dei file, per gli URL, per il testo enfatizzato, e per il primo utilizzo dei termini tecnici.

`Monospazio`
: Un font `monospazio` è usato per i messaggi di errore, i comandi, le variabili di ambiente, i nomi di host, i nomi degli utenti, i nomi dei gruppi, i nomi dei device, le variabili, e i frammenti di codice.

Grassetto
: Un font in grassetto è per le applicazioni, i comandi, e i tasti.

Input dell'Utente

I tasti sono visualizzati in grassetto per differenziarli dal testo normale. Le combinazioni di tasti che devono essere digitate contemporaneamente sono visualizzate con un `+' tra i tasti, come:

Ctrl+Alt+Del

I tasti che devono essere digitati in sequenza saranno separati da virgole, come per esempio:

Ctrl+X, Ctrl+S

Vuol dire che l'utente deve digitare i tasti Ctrl e X contemporaneamente e poi i tasti Ctrl e S.

Capitolo 1. Storia di Unix

1965 Bell Laboratory con la collaborazione del MIT e della General Eletric lavorano per la realizzazione di un nuovo sistema operativo, Multics, il quale vuole fornire, come principali caratteristiche, capacità multi-utente (multi-user), multi-processo (multi-processor) e un file system multi-livello (gerarchico) (multi-level file system).

1969 AT&T era infelice del progresso di Multics e abbandona il progetto. Ken Thompson, Dennis Ritchie, Rudd Canaday e Doug McIlroy, alcuni programmatori dei Bell Lab che avevano lavorato nel progetto Multics, progettano e implementano su un PDP-7 la prima versione del file system Unix insieme ad alcune utility. Il nome Unix è stato assegnato da parte di Brian Kernighan come gioco di parole su Multics.

1 Gennaio 1970 Inizio di Unix.

1971 Il sistema ora gira su un PDP-11 con 16 Kbyte di memoria, di cui 8 Kbyte per i programmi utente, e con un disco di 512 Kbyte.

Il suo primo reale impiego è come strumento di manipolazione del testo in esclusiva per il dipartimento dei Bell Lab. Quel tipo di utilizzo giustifica ulteriormente la ricerca e lo sviluppo attraverso la programmazione di gruppo. Unix attira i programmatori perchè è stato progettato con queste caratteristiche:

- ambiente di programmazione;
- semplice interfaccia utente;
- semplici utility che possono essere combinate per realizzare potenti funzioni;
- file system gerarchico (ad albero);
- semplice interfacciamento con i dispositivi, in armonia con il formato dei file;
- sistema multi-utente e multi-processo;
- architettura indipendente e trasparente all'utente.

1973 Unix è riscritto prevalentemente in C, un nuovo linguaggio di programmazione sviluppato da Dennis Ritchie. La codifica in questo linguaggio di alto livello diminuisce fortemente lo sforzo necessario per portare Unix su nuove macchine.

1974 Thompson e Ritchie descrivono in una relazione pubblicata in un comunicato dell'ACM il nuovo sistema operativo Unix. Unix genera entusiasmo nella comunità accademica che lo vede come un potente strumento di insegnamento per lo studio della programmazione di sistemi. Poichè il decreto del 1956 impedisce ad AT&T di commercializzare il prodotto, viene concessa una licenza all'Università per scopi educativi e una per esistenza commerciale.

1977 Ci sono circa 500 siti Unix nel mondo.

1980 BSD 4.1 (software sviluppato da Berkeley).

1983 SunOS, BSD 4.2, Sys V.

1984 Ci sono circa 100.000 siti Unix che girano su differenti piattaforme hardware, con differenti capacità.

1988 AT&T e Sun Microsystem sviluppano System V Release 4 (SVR4). Questo sarà in futuro implementato in UnixWare e Solaris 2.

1993 Novell compra Unix da AT&T.

1994 Novell porta il nome *UNIX* a X/OPEN.

1995 Santa Cruz Operation compra UnixWare da Novell. Santa Cruz Operation e Hewlett-Packard annunciano lo sviluppo di una versione di Unix a 64 bit.

1996 International Data Corporation prevede che nel 1997 ci saranno 3 milioni di sistemi Unix nel mondo.

Capitolo 2. Struttura di Unix

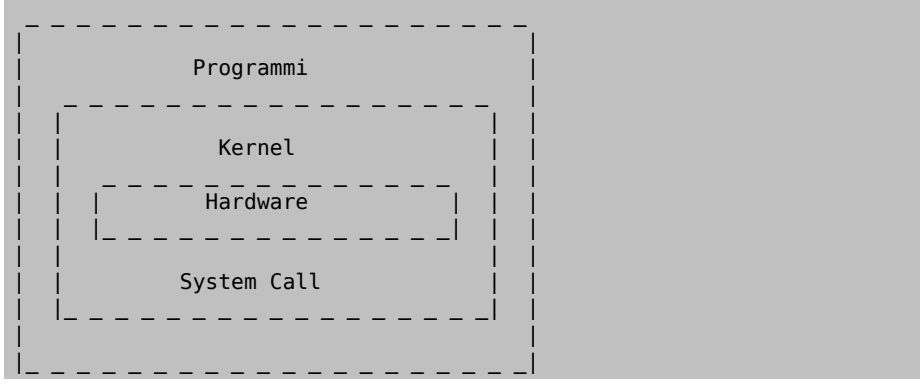
2.1. Il sistema operativo

Unix è un sistema operativo a strati. Lo strato più interno è l'hardware il quale fornisce servizi al OS. Il sistema operativo (OS), riferito in Unix come al *kernel*, interagisce direttamente con l'hardware e fornisce i servizi ai programmi utente. I programmi utente non necessitano di conoscere informazioni sull'hardware. Devono solo sapere come interagire con il kernel ed è quest'ultimo a fornire i servizi richiesti. Uno dei più grandi fattori che ha contribuito alla richiesta di Unix da parte dei programmatori è stato che molti programmi utente corretti sono indipendenti dall'hardware sottostante, e ciò li rende facilmente trasportabili su nuovi sistemi.

I programmi utente interagiscono con il kernel attraverso un set di *system call* (chiamate di sistema) standard. Queste system call chiedono dei servizi, servizi che saranno forniti dal kernel. Così i servizi possono includere un accesso a un file: aprire, chiudere, leggere, scrivere un file, creare un link o eseguire un file; creare o aggiornare degli account (informazioni relative ad un utente come nome, password, ecc.); cambiare il proprietario di un file o di una directory; spostarsi in una nuova directory; creare, sospendere o terminare un processo; abilitare l'accesso a un dispositivo hardware e impostare dei limiti sulle risorse di sistema.

Unix è un sistema operativo *multi-user* (multi-utente) e *multi-tasking* (multi-processo). Si possono avere molti utenti «loggati» simultaneamente nel sistema (multi-user), ognuno dei quali esegue alcuni programmi (multi-tasking). È compito del kernel mantenere ogni processo e ogni utente separato e regolare l'accesso all'hardware di sistema, inclusa la cpu, la memoria, il disco e altri dispositivi di I/O.

FIGURA 2.1 Struttura di un sistema Unix



2.2. Il file system

L'aspetto del file system di Unix è paragonabile alla struttura rovesciata di un albero. Si parte dall'alto con la directory *root*, denotata con `/`, per poi scendere attraverso sotto-directory sottostanti la root.

```
FIGURA 2.2   Struttura del file system di Unix


                       /(root)                        1 livello
         |           |         |         |     |
       bin         dev         etc      tmp    usr    2 livello
    |  |  |        | |         | |             | |
  sh date csh  ttya cua0  passwd group       bin lib  3 livello
                                              |
                                             gcc       4 livello

                        ....                           n livello
```

Ogni nodo è o un *file* o una *directory* di file, dove quest'ultima può contenere altri file e directory. Un file o una directory vengono specificati attraverso il loro *path name* (percorso del nome del file o della directory), il quale può essere un path name assoluto oppure un path name relativo ad un'altra locazione. Un path name assoluto inizia con la directory root, `/`, seguono poi i «rami» del file system, ognuno separato da `/`, fino a raggiungere il file desiderato, come per esempio:

```
/home/condron/source/xntp
```

Un path name relativo specifica un percorso relativo ad un altro path name, che usualmente è la directory di lavoro corrente in cui ci si trova. Sono ora introdotte due directory speciali:

`.` la directory corrente

`..` la directory padre della directory corrente

Quindi se si è in `/home/frank` e si desidera specificare un path nel modo relativo si può usare:

```
../condron/source/xntp
```

Questo indica che si deve prima salire di un livello di directory, quindi passare attraverso la directory `condron`, seguire la directory `source` e quindi `xntp`.

2.3. Directory, file e inode Unix

Ogni *directory* e ogni *file* sono inclusi nella loro directory padre. Nel caso della directory root, la directory padre è se stessa. Una directory è un file contenente una tabella che elenca i file contenuti nella directory stessa, dove ai nomi dei file in lista vengono assegnati i corrispondenti numeri di *inode*. Un inode è un file speciale, progettato per essere letto dal kernel al fine di conoscere alcune informazioni su ciascun file. Un inode specifica i permessi del file, il proprietario del file, la data di creazione, quella dell'ultimo accesso e quella dell'ultima modifica del file e la posizione fisica dei blocchi di dati sul disco che contengono il file.

Il sistema non richiede qualche struttura particolare per i dati contenuti nel file. Il file può essere ASCII o binario o una combinazione di questi e può rappresentare dati testuali, uno script di shell, un codice oggetto compilato per un programma, una tabella di directory, robaccia o qualunque cosa si voglia.

Non c'è un'intestazione, una traccia, un'etichetta o il carattere *EOF* come parte del file.

2.4. Programmi Unix

Un *programma* o un *comando* interagisce con il kernel per fornire l'ambiente e realizzare le funzioni richieste dall'utente. Un programma può essere: un file di shell eseguibile, conosciuto come uno script di shell, un comando interno (built-in) alla shell o un file sorgente compilato in codice oggetto.

La *shell* è un interprete a linea di comando. L'utente interagisce con il kernel attraverso la shell. Si può scrivere uno script ASCII (testo) in modo tale da essere interpretato da una shell.

I programmi di sistema sono generalmente in forma binaria, compilati partendo da un codice sorgente in C. Questi si trovano in posti come `/bin`, `/usr/bin` , `/usr/local/bin` , `/usr/ucb` , ecc. Questi comandi forniscono quelle funzionalità che normalmente si pensano essere di Unix. Alcuni di questi sono sh(1), csh(1), date(1), who(1), more(1), ls(1) e molti altri.

Capitolo 3. Iniziamo

3.1. Effettuare il login

Una volta che l'utente si è collegato a un sistema Unix, gli viene chiesto di inserire un *login* username (nome utente) e una *password* (codice segreto). Il login username è il nome univoco dell'utente sul sistema. La password è un codice modificabile conosciuto solo dall'utente. Alla richiesta di *login*, l'utente deve inserire lo username e alla richiesta della *password*, deve essere inserita la password effettiva.

Nota

Unix è un sistema case sensitive (fa distinzione tra caratteri minuscoli e maiuscoli). Quindi sia il *login* username che la *password* devono essere inseriti esattamente come sono stati creati; il login username è normalmente in minuscolo.

3.1.1. Tipi di terminali

Molti sistemi sono configurati in modo tale da richiedere all'utente il tipo di terminale da usare, il quale dovrebbe essere settato al tipo di terminale usato nella fase di login. Molti computer lavorano se si sceglie il tipo `vt100`. Gli utenti connessi tramite una workstation Sun molto probabilmente useranno il tipo `sun`; quelli che usano un Terminale-X molto probabilmente useranno `xterms` oppure `xterm`.

Il tipo di terminale indica al sistema Unix come interagire quando una sessione viene aperta.

Può essere necessario risettare il tipo di terminale, digitando il comando:

`setenv TERM <tipo di terminale>`	-se si usa la shell C (vedere Capitolo 5)

(Inoltre su alcuni sistemi Unix, ad esempio MAGNUS, è necessario dare il comando `unsetenv TERMCAP`.)

oppure

`TERM=<tipo di terminale>; export TERM`	-se si usa la shell Bourne (vedere Capitolo 5)

dove <*tipo di terminale*> è il tipo di terminale, come `vt100`, che si desidera settare.

3.1.2. Password

Quando viene assegnato il proprio account, si riceve una password iniziale. Questa password è importate sia per la sicurezza del sistema sia per una sicurezza personale, perciò la password per il proprio account può essere cambiata a propria scelta. Il comando per cambiare una password è passwd(1). Verrà chiesto di inserire sia la propria vecchia password, sia la nuova password desiderata, quest'ultima due volte. Se si sbaglia la propria vecchia password o non si inserisce quella nuova nello stesso modo per due volte, il sistema indicherà che la password non ha subito cambiamenti.

Alcuni amministratori di sistema hanno programmi che verificano l'accuratezza di una password (che deve essere sufficientemente criptica per assicurare una certa sicurezza al sistema). Un cambiamento di password può essere rifiutato da questi programmi.

Quando si sceglie una password, è importante che questa non sia in qualche modo indovinabile né per qualche sconosciuto che tenta di scoprirla né per un conoscente. I suggerimenti per scegliere e mantenere una password sono i seguenti:

- non usare una parola (o parole) di un linguaggio;
- non usare il proprio nome;
- non usare informazioni che possono essere trovate nel proprio portafoglio;
- non usare informazioni comunemente conosciute circa se stessi (come il soprannome, patente di guida, ecc.);
- non usare caratteri di controllo. Alcuni sistemi non li accettano;
- non scrivere la password in qualche posto;
- non dare la propria password a *nessuno*;
- usare un mix di tipi di caratteri (alfabetici, numerici, speciali);
- usare un mix di caratteri maiuscoli e caratteri minuscoli;
- usare un minimo di 6 caratteri;
- scegliere una password in modo da poterla ricordare;
- cambiare di frequente la propria password;
- assicurarsi che nessuna persona vicino a voi vi guardi quando si inserisce la propria password.

3.1.3. Uscita

Ctrl+D - indica la fine di un flusso di dati; può far uscire dal sistema un utente. L'ultimo caso è disabilitato su molti sistemi

Ctrl+C - interruzione

logout(1) - rilascia il sistema

exit(1) - rilascia la shell

3.1.4. Identità

Il sistema identifica un utente attraverso il numero di utente e il numero di gruppo (rispettivamente *userid* e *groupid*) assegnati dall'amministratore di sistema. Generalmente non è necessario conoscere il proprio userid e groupid, poichè il sistema traduce in modo automatico lo userid in username (e viceversa) ed il groupid in groupname (e viceversa). Probabilmente si conosce già lo username: è il nome utilizzato per il login. Il groupname non è ovvio, e in realtà si può appartenere a più di un gruppo. Il proprio gruppo primario è quello associato con il proprio username nel file database delle password, configurato dall'amministratore di sistema. Similmente, c'è un file database per i gruppi, dove l'amministratore di sistema può assegnare gruppi aggiuntivi ad un utente.

Nei seguenti esempi il simbolo `%` è il proprio prompt di shell e non va digitato.

Si può determinare il proprio userid e la lista dei gruppi di appartenenza con i comandi id(1) e groups(1). Su alcuni sistemi id(1) mostra le informazioni sull'utente e le informazioni sul gruppo primario, esempio:

```
% id
uid=1101(frank) gid=10(staff)
```

su altri sistemi mostra anche le informazioni sui gruppi aggiuntivi di appartenenza:

```
% id
uid=1101(frank) gid=10(staff) groups=10(staff),5(operator),14↺
(sysadmin),110(uts)
```

Il comando groups(1) mostra le informazioni di tutti i gruppi di appartenenza, esempio:

```
% groups
staff sysadmin uts operator
```

3.2. Struttura della linea di comando di Unix

Un *comando* è un programma che chiama il sistema Unix per qualche compito. Un comando ha la forma:

```
comando [opzioni] [argomenti]
```

dove un *argomento* indica su cosa il comando deve realizzare la sua azione, generalmente un file o una serie di file. Un'opzione modifica il comando, cambiandone il modo di esecuzione.

I comandi sono case sensitive (sensibili alle lettere maiuscole e minuscole). `comando` e `Comando` non sono la stessa cosa.

Le *opzioni* sono generalmente precedute da un trattino (-) e per molti comandi, più opzioni possono essere messe insieme nella forma:

```
comando -[opzione][opzione][opzione]
```

esempio:

```
ls -alR
```

che mostrerà un listato lungo di tutti i file che si trovano nella directory corrente e ricorsivamente anche quelli che si trovano in tutte le sotto-directory.

In molti comandi si possono separare le opzioni, facendole precedere ognuna da un trattino, esempio:

```
comando -opzione1 -opzione2 -opzione3
```

come in:

```
ls -a -l -R
```

Alcuni comandi hanno opzioni che richiedono parametri. Le opzioni che richiedono parametri sono di solito specificate separatamente, esempio:

```
lpr -Pprinter3 -#2 file
```

che trasmetterà 2 copie del file specificato a `printer3`.

Ci sono delle convenzioni standard per i comandi. Comunque, non tutti i comandi Unix seguono questo standard. Alcuni non richiedono il trattino prima dell'opzione e alcuni non permettono di mettere insieme più opzioni, per esempio alcuni possono richiedere che ogni opzione sia preceduta da un trattino e separata con uno spazio bianco da un'altra opzione o argomento.

Le opzioni e la sintassi di un comando sono mostrate nelle *pagine man* del comando.

3.3. Tasti di controllo

I *tasti di controllo* sono usati per realizzare speciali funzioni su linea di comando o all'interno di un editor. Queste funzioni possono essere generate premendo contemporaneamente il tasto control e alcuni altri *tasti*. Questa combinazione è generalmente indicata con Ctrl+***Tasto*** (oppure ^+***Tasto***). Control+S può essere scritto come Ctrl+S (oppure ^+S). Con i tasti di controllo le lettere maiuscole e minuscole sono la stessa cosa, così

Ctrl+S è lo stesso di Ctrl+s. Questo particolare esempio (Ctrl+S) è un segnale di *stop* e dice al terminale di non accettare più input. Il terminale rimarrà sospeso finchè un segnale di *start* Ctrl+Q non sarà generato.

Ctrl+U è normalmente il segnale di «*cancellazione di linea*» per il proprio terminale. Quando lo si digita, l'intera linea di input viene cancellata.

Nell'editor vi(1) si possono inserire i tasti di controllo all'interno del file di testo facendo seguire a Ctrl+V il carattere di controllo desiderato; così per inserire in un documento Ctrl+H si digita Ctrl+V, Ctrl+H.

3.4. stty - controllo del terminale

stty(1) mostra o configura le opzioni di controllo del terminale. L'abbreviazione «*tty*» risale fino ai giorni dei «teletypewrite», che erano associati alla trasmissione di messaggi telegrafici ed erano primitivi modelli di terminali di computer.

Per i nuovi utenti, l'uso principale del comando stty(1) riguarda l'assegnazione della funzione di «cancellazione di linea» ad un tasto specifico per i loro terminali. Per i programmatori di sistema o per chi scrive script di shell, il comando stty(1) fornisce uno strumento prezioso per la configurazione di molti aspetti legati al controllo di I/O di un dispositivo specifico, inclusi i seguenti:

- carattere di erase (eliminazione carattere) e di line-kill (eliminazione linea);
- velocità di trasmissione dati;
- controllo di parità sulla trasmissione dati;
- controllo del flusso hardware;
- carattere di nuova linea (<NL>), di return (<CR>) e di alimentazione linea (<LF>);
- interpretazione del carattere tab;
- modifica di un input grezzo;
- trasformazione di lettere minuscole in lettere maiuscole.

Il comando stty(1) è molto dipendente dal sistema, quindi consultare le relative *pagine man* sul proprio sistema per i dettagli.

Sintassi

```
stty [opzioni]
```

Opzioni generali

`(none)`	mostra i settaggi del terminale
`all (-a)`	mostra tutte le opzioni
`echoe`	richiama ERASE come BS-spazio-BS
`dec`	imposta la modalità specifica dei sistemi operativi conformi al Digital Equipment Corporation (che distinguono ERASE da BACKSPACE) (non disponibile su tutti i sistemi)
`kill`	imposta il carattere per LINE-KILL
`erase`	imposta il carattere per ERASE
`intr`	imposta il carattere per INTERRUPT

Esempi:

Con il comando stty(1) si possono visualizzare e cambiare i settaggi del proprio terminale di controllo. Per visualizzare tutti (`-a`) i settaggi correnti:

```
% stty -a
speed 38400 baud, 24 rows, 80 columns
parenb -parodd cs7 -cstopb -hupcl cread -clocal -crtscts
-ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr icrnl -
iuclc
ixon -ixany -ixoff imaxbel
isig iexten icanon -xcase echo echoe echok -echonl -noflsh -tostop
echoctl -echoprt echoke
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel
erase kill werase rprnt flush lnext susp intr quit stop eof
^H     ^U    ^W     ^R    ^O   ^V  ^Z/^Y  ^C   ^\  ^S/^Q ^D
```

Per cambiare i settaggi usando stty(1), ad esempio per cambiare il carattere di erase da Ctrl+? (il tasto elimina) a Ctrl+H:

```
% stty erase ^H
```

Questo setterà l'opzione del terminale solamente per la sessione corrente. Per far in modo che questo comando sia eseguito automaticamente ad ogni login, è possibile inserire tale comando nel file `.login` o `.profile` di cui si parlerà più avanti.

3.5. Ottenere aiuto

Il manuale di Unix, usualmente chiamato *man page* (pagine man), è disponibile per spiegare l'uso del sistema Unix e dei suoi comandi. Per servirsi di una pagina man digitare il comando man(1) al prompt di sistema seguito dal comando di cui si necessitano informazioni.

Sintassi

```
man [opzioni] nome_comando
```

Opzioni generali

`-k` *`parola_chiave`*	mostra alcune linee riassuntive dei comandi contenenti la parola chiave richiesta
`-M` *`path`*	percorso per le pagine man
`-a`	mostra tutte le pagine man trovate (SVR4)

Esempi:

Si può usare man(1) per ottenere una linea riassuntiva di alcuni comandi che contengono la parola che si vuole ricercare con l'opzione `-k`, ad esempio per cercare la parola *password*, si digita:

```
% man -k password
passwd (5) - password file
passwd (1) - cambia la password
```

Il numero in parentesi indica la sezione delle pagine man dove sono stati trovati i riferimenti. Si può accedere a quella pagina man (di default si fa riferimento al numero di sezione più basso, ma si può usare un'opzione su linea di comando per specificarne uno differente) con:

```
% man passwd
PASSWD(1) USER COMMANDS PASSWD(1)

NOME
passwd - cambia password

SINTASSI
passwd [ -e login_shell -] [ username -]

DESCRIZIONE
passwd cambia (o setta) la password di un utente.
passwd chiede per due volte la nuova password, senza mostrarla.
Questo per prendere in considerazione la possibilità di digitare ↺
errori.
Solamente l'utente stesso e il super-user possono cambiare la ↺
password
di un utente.

OPZIONI

-e Cambia la shell di login dell'utente.
```

Qui l'output è stato parafrasato e troncato per una questione di spazio e di copyright.

3.6. Navigazione e controllo delle directory

Il file system di Unix è organizzato come la struttura ramificata di un albero a partire da root. La directory *root* del sistema è rappresentata dal carattere di slash in avanti (/). Le directory di sistema e quelle degli utenti sono organizzate sotto la directory *root*. In Unix l'utente non ha una directory root; generalmente dopo il login gli utenti vengono posizionati nella loro directory *home*. Gli utenti possono creare altre directory sotto la loro directory *home*. La tabella che segue mostra alcuni comandi per la navigazione tra directory.

Tabella 3.1. Comandi di navigazione e controllo delle directory

Comando/Sintassi	Cosa fa
`cd [directory]`	cambia directory
`ls [opzioni][directory o file]`	lista il contenuto della directory o i permessi del file specificato
`mkdir [opzioni] directory`	crea una directory
`pwd`	mostra la directory (corrente) di lavoro
`rmdir [opzioni] directory`	rimuove una directory

Se si ha una certa familiarità con DOS la tabella che segue paragona i suoi simili comandi a quelli Unix in modo tale da fornire un appropriato quadro di riferimento.

Tabella 3.2. Comandi di navigazione e controllo delle directory Unix vs DOS

Comando	Unix	DOS
lista il contenuto di una directory	`ls`	`dir`
crea una directory	`mkdir`	`md` & `mkdir`
cambia directory	`cd`	`cd` & `chdir`
rimuove una directory	`rmdir`	`rm` & `rmdir`
ritorna alla directory home dell'utente	`cd`	`cd \`

Comando	Unix	DOS
mostra la directory corrente di lavoro	`pwd`	`cd`

3.6.1. pwd - mostra la directory di lavoro

In ogni momento si può determinare in che punto si è nella gerarchia del file system mostrando la directory di lavoro con il comando pwd(1), esempio:

```
% pwd
/home/frank/src
```

3.6.2. cd - cambia directory

Ci si può portare in una nuova directory con il comando cd(1), cambio di directory. cd(1) accetta sia path name (percorsi) assoluti sia path name relativi.

Sintassi

```
cd [directory]
```

Esempi:

`cd` (oppure `chdir` in alcune shell)	cambia directory
`cd`	si posiziona nella directory home dell'utente
`cd /`	si posiziona nella directory di sistema roor (/)
`cd ..`	sale di un livello di directory
`cd ../..`	sale di due livelli di directory
`cd /completo/path/name/da/root`	cambia directory rispetto a un path name assoluto (notare lo slash iniziale)
`cd path/da/posizione/corrente`	cambia directory rispetto a un path name relativo alla posizione corrente (no slash iniziale)
`cd ~username/directory`	cambia directory rispetto alla directory home dell'utente specificato (il carattere ~ non è valido nella shell Bourne; vedere il Capitolo 5).

3.6.3. mkdir - crea una directory

La gerarchia della propria directory home si estende creando sotto-directory all'interno di essa. Questo è possibile con il comando mkdir(1), crea directory. Di nuovo si può specificare un path name assoluto o relativo della directory che si vuole creare.

Sintassi

```
mkdir [opzioni] directory
```

Opzioni generali

`-p`	crea una directory intermedia (genitore), quando necessario
`-m` *modi*	permessi di accesso (SVR4). (Si vedranno i «modi» più avanti in questo Capitolo)

Esempi:

```
% mkdir /home/frank/data
```

oppure se la directory di lavoro corrente è `/home/frank` , il seguente comando è equivalente:

```
% mkdir data
```

3.6.4. rmdir - rimuove una directory

Per rimuovere una directory è necessario che questa sia vuota. Altrimenti bisogna prima rimuovere i file contenuti in essa. Inoltre, non si può rimuovere una directory se questa è la directory di lavoro corrente, bisogna prima uscire da quest'ultima.

Sintassi

```
rmdir directory
```

Esempi:

Per rimuovere la directory vuota `/home/frank/data` mentre si è in `/home/frank` usare:

```
% rmdir data
```

oppure

```
% rmdir /home/frank/data
```

3.6.5. ls - mostra i contenuti delle directory

Il comando per visualizzare le proprie directory e i propri file è ls(1). È possibile ottenere, attraverso le opzioni, informazioni circa la dimensione, il tipo, i permessi, la data di creazione, di modifica e di accesso del file.

Sintassi

```
ls [opzioni] [argomenti]
```

Opzioni generali

Quando non viene usato nessun argomento, viene mostrato il contenuto della directory corrente. Ci sono molte utili opzioni per il comando ls(1). Segue una lista di alcune di queste. Quando si usa il comando, le opzioni sono raggruppate insieme, precedute da un trattino (-).

`-a`	mostra tutti i file, inclusi quelli che iniziano con un punto (.)
`-d`	mostra solo i nomi delle directory, non i file nella directory
`-F`	indica il tipo di elemento terminandolo con un simbolo:
	directory /
	socket =
	link simbolico @
	eseguibile *
`-g`	mostra il gruppo Unix assegnato al file, richiede l'opzione `-l` (BSD solamente) o su una macchina SVR4, esempio Solaris, questa opzione ha l'effetto opposto
`-L`	se il file è un link simbolico, mostra le informazioni del file o della directory a cui il link si riferisce, non le informazioni del link stesso
`-l`	listato lungo: mostra i modi, informazioni di link, il proprietario, la dimensione, la data dell'ultima modifica del file. Se il file è un link simbolico, una freccia (-->) precede il percorso del file collegato.

Il *campo modi* viene fornito dall'opzione `-l` e consiste di 10 caratteri. Il primo carattere è uno dei seguenti:

CARATTERE	SE L'ELEMENTO E'
d	directory
-	file ordinario
b	file speciale per dispositivi a blocchi
c	file speciale per dispositivi a caratteri
l	link simbolico
s	socket

I 9 caratteri successivi sono raggruppati in 3 blocchi di 3 caratteri ciascuno. Indicano i *permessi di accesso al file*: i primi 3 caratteri si riferiscono ai permessi del *proprietario* del

file, i successivi 3 ai permessi degli utenti del *gruppo* Unix assegnato al file e gli ultimi 3 caratteri ai permessi degl'*altri* utenti sul sistema. Possono assumere i seguenti simboli:

r	permesso di lettura
w	permesso di scrittura
x	permesso di esecuzione
-	permesso negato

Esistono altri permessi, specificamente progettati per essere usati in speciali situazioni. Questi sono spiegati nelle pagine man di ls(1).

Esempi:

Per mostrare i file in una directory

```
% ls
demofiles  frank  linda
```

Per mostrare tutti i file in una directory, inclusi i file nascosti (iniziano con un punto):

```
% ls -a
.   .cshrc  .history  .plan     .rhosts    frank
..  .emacs  .login    .profile  demofiles  linda
```

Per avere un listato lungo:

```
% ls -al
total 24
drwxr-sr-x 5 workshop acs 512 Jun 7 11:12  .
drwxr-xr-x 6 root     sys 512 May 29 09:59 ..
-rwxr-xr-x 1 workshop acs 532 May 20 15:31 .cshrc
-rw------- 1 workshop acs 525 May 20 21:29 .emacs
-rw------- 1 workshop acs 622 May 24 12:13 .history
-rwxr-xr-x 1 workshop acs 238 May 14 09:44 .login
-rw-r--r-- 1 workshop acs 273 May 22 23:53 .plan
-rwxr-xr-x 1 workshop acs 413 May 14 09:36 .profile
-rw------- 1 workshop acs 49  May 20 20:23 .rhosts
drwx------ 3 workshop acs 512 May 24 11:18 demofiles
drwx------ 2 workshop acs 512 May 21 10:48 frank
drwx------ 3 workshop acs 512 May 24 10:59 linda
```

3.7. Comandi di gestione dei file

Per creare, copiare, rimuovere file e per modificarne i permessi si possono usare i seguenti comandi.

Tabella 3.3. Comandi di gestione dei file

Comando/Sintassi	Cosa fa
`chgrp [opzioni] gruppo file`	cambia il gruppo assegnato ad un file
`chmod [opzioni] file`	cambia i permessi di accesso a file o directory
`chown [opzioni] proprietario file`	cambia il proprietario di un file; può essere usato solamente dal super-user
`cp [opzioni] file1 file2`	copia `file1` in `file2`; `file2` non dovrebbe già esistere. Questo comando crea o sovrascrive `file2`
`mv [opzioni] file1 file2`	muove (rinomina) `file1` in `file2`
`rm [opzioni] file`	elimina un file o una directory (`-r` rimuove ricorsivamente le directory e il loro contenuto) (`-i` chiede conferma prima di rimuove i file)

Se si ha una certa familiarità con DOS la tabella che segue paragona i suoi simili comandi a quelli Unix in modo tale da fornire un appropriato quadro di riferimento.

Tabella 3.4. Comandi di gestione dei file Unix vs DOS

Comando	Unix	Dos
copia un file	`cp`	`copy`
muove un file	`mv`	`move` (non supportato in tutte le versio-

Comando	Unix	Dos
		ni di Dos)
rinomina un file	`mv`	`re-na-me` & `ren`
elimina un file	`rm`	`era-se` & `del`
mostra un file a schermo	`cat`	`ty-pe`
mostra un file a schermo, una pagina alla volta	`more`, `less`, `pg`	`ty-pe` / `p` (non sup-por-ta-to in tut-te le ver-sio-ni di Dos)

3.7.1. cp - copia un file

Il comando cp(1) copia il contenuto di un file in un altro file.

Sintassi

```
cp [opzioni] filename1 filename2
```

Opzioni generali

`-i`	interattivo (chiede conferma prima di procedere)

`-r`	copia ricorsivamente una directory

Esempi:

```
% cp filename1 filename2
```

Si hanno due copie del file, ognuna con un identico contenuto. Questi file sono completamente indipendenti tra loro e possono essere editati e modificati entrambi quando necessario. Ciascuno di essi ha il proprio inode, i propri blocchi di dati e il proprio elemento nella tabella di directory.

3.7.2. mv - sposta un file

Il comando mv(1) rinomina (sposta) un file.

Sintassi

```
mv [opzioni] vecchio_file nuovo_file
```

Opzioni generali

`-i`	interattivo (chiede conferma prima di procedere)
`-f`	non chiede la conferma quando si sovrascrive un file esistente (ignora `-i`)

Esempi:

```
% mv vecchio_file nuovo_file
```

Il file `nuovo_file` sostituisce `vecchio_file` . In realtà tutto quello che è stato fatto è aver aggiornato l'elemento della tabella di directory per attribuire al file il nuovo nome. Il contenuto del file rimane come era prima della rinominazione.

3.7.3. rm - elimina un file

Il comando rm(1) elimina un file.

Sintassi

```
rm [opzioni] filename
```

Opzioni generali

`-i`	interattivo (chiede conferma prima di procedere)
`-r`	rimuove una directory ricorsivamente, rimuovendo prima i file e le directory sottostanti
`-f`	non chiede conferma prima di procedere (ignora `-i`)

Esempi:

```
% rm filename
```

Visualizzando il contento della directory si vedrà che quel file non esiste più. In realtà tutto quello che è stato fatto è aver rimosso l'elemento dalla tabella di directory e marcato l'inode come «non usato». Il contenuto del file è ancora sul disco, ma ora il sistema non ha più modo di identificare quei blocchi di dati con il nome del file eliminato. Non c'è un certo comando per «*riprendere*» un file che è stato eliminato in questo modo. Per questa ragione molti utenti alle prime armi effettuano un alias del comando di eliminazione in `rm -i` dove l'opzione `-i` chiede di confermare prima di rimuovere il file. Simili alias sono generalmente messi nel file `.cshrc` per la shell C (vedere il Capitolo 5).

3.7.4. Permessi dei file

Ciascun file e directory ha permessi che stabiliscono chi può *leggerlo*, *scriverlo* e/o *eseguirlo*. Per scoprire i permessi assegnati a un file, può essere usato il comando ls(1) con l'opzione `-l`. Quando si ha la necessità di conoscere il gruppo per il quale i permessi sono stati assegnati si può usare l'opzione `-g` insieme a `ls -l` (solamente BSD).

Quando si usa il comando `ls -lg` su un file (`ls -l` su SysV) l'output sarà mostrato come il seguente:

```
-rwxr-x--- user Unixgroup size Month nn hh:mm filename
```

La zona dedicata ai caratteri e trattini (*-rwxr-x---*) è la zona che mostra il tipo di file e i permessi del file, come spiegato nella precedente sezione. Quindi la stringa di permessi dell'esempio, *-rwxr-x---*, permette al proprietario `user` del file di leggerlo, modificarlo ed eseguirlo; gli utenti del gruppo `Unixgroup` del file possono leggerlo ed eseguirlo; gli *altri* utenti del sistema non possono accedere in alcun modo al file.

3.7.5. chmod - cambio dei permessi del file

Il comando per cambiare i permessi ad un elemento (file, directory, ecc.) è chmod(1) (cambio dei modi). La sintassi richiede l'uso del comando con tre cifre (rappresentanti i permessi del *proprietario* (*u*), i permessi del *gruppo* (*g*) e i permessi degli *altri* utenti (*o*)) seguite da un argomento (che può essere un nome di un file o una lista di file e directory). Oppure può essere usato con una rappresentazione simbolica dei permessi, indicando a quale utenza questi vanno applicati.

Ogni tipo di permesso è rappresentato dal proprio numero equivalente:

lettura=4, scrittura=2, esecuzione=1

o da singoli caratteri:

lettura=r, scrittura=w, esecuzione=x

Il permesso *4* o *r* specifica il permesso di *lettura*. Se i permessi desiderati sono lettura e scrittura, il 4 (rappresentante la lettura) e il 2 (rappresentante la scrittura) sono addizionati per ottenere il permesso 6. Quindi, un permesso settato a 6 vorrà concedere un permesso di lettura e di scrittura.

Alternativamente si può usare una notazione simbolica che usa un carattere rappresentativo per l'utenza a cui ci si riferisce, uno per il permesso e uno per l'operazione, dove l'operazione può essere:

`+`	aggiunge permessi
`-`	rimuove permessi
`=`	setta permessi

Quindi per settare i permessi di lettura e di scrittura per il proprietario del file si usa nella notazione simbolica *u=rw*.

Sintassi

`chmod nnn [lista argomenti]`	modalità numerica
`chmod [chi] op [perm] [lista argomenti]`	modalità simbolica

dove *nnn* sono i tre numeri rappresentanti i permessi del *proprietario*, del *gruppo* e degli *altri* utenti; *`chi`* può essere *u,g,o* oppure *a* (tutti) e *`perm`* può essere *r,w,x*. Nella notazione simbolica si può separare la specifica dei permessi con una virgola, come mostrato nell'esempio qui sotto.

Opzioni generali

`-f`	forza (nessun messaggio di errore viene generato se la modifica non ha avuto successo)
`-R`	discesa ricorsiva attraverso la struttura delle directory e cambio dei modi

Esempi:

Se i permessi desiderati per il `file1` sono: *proprietario*: lettura, scrittura ed esecuzione; *gruppo*: lettura ed esecuzione; *altri*: lettura ed esecuzione; il comando da usare è:

`chmod 755 file1` oppure `chmod u=rwx,go=rx file1`

Nota

Quando si assegnano i permessi a un file per l'utenza *gruppo* e per l'utenza *altri* è necessario che il minimo permesso di esecuzione (inteso come permesso di accesso) per la directory nella quale il file

è posizionato sia abilitato. Un semplice modo per far questo è posizionarsi nella directory nella quale i permessi devo essere garantiti e digitare:

`chmod 711 .` oppure `chmod u=rw,+x .` oppure `chmod u=rwx,go=x .`

dove il punto (.) indica la *directory corrente*.

3.7.6. chown - cambio del proprietario del file

Il proprietario di un file può essere cambiato con il comando chown(8). Su molte versioni Unix questo può essere realizzato solo dal super-user, ad esempio, un utente normale non può attribuire i suoi file ad altri proprietari. chown(8) è usato come qui sotto, dove # rappresenta il prompt di shell per il super-user.

Sintassi

`chown [opzioni] utente[:gruppo] file`	(SVR4)
`chown [opzioni] utente[.gruppo] file`	(BSD)

Opzioni generali

`-R`	discende ricorsivamente attraverso la struttura della directory
`-f`	forza, non riporta errori

Esempi:

```
# chown nuovo_proprietario file
```

3.7.7. chgrp - cambio del gruppo del file

Con il comando chgrp(1) tutti possono cambiare il gruppo dei propri file in un altro gruppo di appartenenza.

Sintassi

```
chgrp [opzioni] gruppo file
```

Opzioni generali

`-R`	discende ricorsivamente attraverso la struttura della directory
`-f`	forza, non riporta errori

Esempi:

```
% chgrp nuovo_gruppo file
```

3.8. Comandi di visualizzazione

Ci sono alcuni comandi che si possono usare per *visualizzare* o *esaminare* un file. Alcuni di questi sono editor che verranno trattati più avanti. Qui si illustreranno alcuni comandi normalmente usati per visualizzare un file.

Tabella 3.5. Comandi di visualizzazione

Comando/Sintassi	Cosa fa
`echo [stringa]`	riporta in standard ouptut una stringa di testo
`cat [opzioni] file`	concatena (lista) un file
`more` (o `less` o `pg`) `[opzioni] file`	visualizzazione paginata di un file di testo
`head [-numero] file`	visualizza le prime 10 (o *`-numero`*) linee di un file
`tail [opzioni] file`	visualizza le ultime linee (o parte) di un file

3.8.1. echo - mostra un'espressione

Il comando echo(1) viene utilizzato per ripetere l'argomento assegnato al comando nel dispositivo standard di uscita. Normalmente l'argomento termina con un carattere di alimentazione di linea, ma si può specificare un'opzione per impedirlo.

Sintassi

```
echo [stringa]
```

Opzioni generali

`-n`	non stampa new-line (BSD, shell built-in)
`\c`	non stampa new-line (SVR4)
`\0`*`n`*	dove *n* è il codice del carattere ASCII a 8 bit (SVR4)
`\t`	tab (SVR4)
`\f`	form-feed (emissione carta) (SVR4)
`\n`	new-line (SVR4)
`\v`	tab verticale (SVR4)

Esempi:

```
% echo Hello Class
```

oppure

```
% echo "Hello Class"
```

Per impedire il carattere di fine linea:

```
% echo -n Hello Class
```

oppure

```
% echo "Hello Class \c"
```

dove il modo utilizzato nell'ultimo esempio dipende dal comando echo(1) usato.

L'opzione *\x* deve essere interna a un paio di caratteri di quoting singoli o doppi, con o senza altri caratteri di stringa.

3.8.2. cat - concatena un file

Il comando di concatenazione cat(1) visualizza il contenuto di un file.

Sintassi

```
cat [opzioni] [file]
```

Opzioni generali

`-n`	precede ogni linea con un numero
`-v`	visualizza i caratteri non stampabili, eccetto tab, new-line e form-feed
`-e`	visualizza `$` alla fine di ogni linea (prima di new-line) (quando usato con l'opzione `-v`)

Esempi:

```
% cat filename
```

Si possono specificare una serie di file su linea di comando e cat(1) li concatenerà ciascuno a turno, seguendo lo stesso ordine di immissione, esempio:

```
% cat file1 file2 file3
```

3.8.3. more, less e pg - visualizzazione paginata di un file

more(1), less(1) e `pg` permettono di visualizzare il contenuto di un file una schermata (pagina) alla volta. Inoltre permettono di ritornare sulla precedente pagina, di cercare parole, ecc. Questi comandi potrebbero non essere disponibili sul proprio sistema Unix.

Sintassi

```
more [opzioni] [+/schema] [filename]

less [opzioni] [+/schema] [filename]

pg [opzioni] [+/schema] [filename]
```

Opzioni

more	less	pg	Azione
`-c`	`-c`	`-c`	pulisce lo schermo prima di visualizzare
	`-i`		ignora differenza tra maiuscole e minuscole
`-w`	default	default	non esce alla

more	less	pg	Azione
			fine dell'input, ma si mette in attesa di comandi
-linee		*-linee*	# (numero di) linee di avanzamento
+/schema	*+/schema*	*+/schema*	ricerca lo schema (pattern)

Controlli interni

`more`	visualizza (una schermata alla volta) il file specificato
<spazio>	per vedere la schermata successiva

<return>o<CR>	per avanzare di una linea
q	per uscire
h	help
b	torna alla schermata precedente
/parola	cerca *parola* nel resto del file
	vedere le *pagine man* per altre opzioni
`less`	simile a more(1), vedere le *pagine man* per le opzioni
`pg`	in SVR4 equivale a more(1) (pagina)

3.8.4. head - mostra l'inizio di un file

Il comando head(1) visualizza l'inizio di un file.

Sintassi

```
head [opzioni] file
```

Opzioni generali

`-n` *numero*	numero di linee da visualizzare partendo dall'inizio del file
-numero	come sopra

Esempi:

Di default head(1) mostra le prime 10 linee del file. Si possono visualizzare più (o meno) linee con l'opzione `-n` *numero* o *-numero*, ad esempio, per visualizzare le prime 40 linee:

```
% head -40 filename
```

oppure

```
% head -n 40 filename
```

3.8.5. tail - mostra la fine di un file

Il comando tail(1) visualizza la fine di un file.

Sintassi

```
tail [opzioni] file
```

Opzioni generali

-numero	numero di linee da visualizzare, partendo dalla fine del file

Esempi:

Di default tail(1) mostra le ultime 10 linee del file, ma si può specificare un numero differente di linee o di byte, o un differente punto di inizio all'interno del file. Per visualizzare le ultime 30 linee di un file, usare l'opzione *-numero*:

```
% tail -30 filename
```

Capitolo 4. Risorse di sistema e stampa

4.1. Risorse di sistema

Comandi per gestire le risorse di sistema.

Tabella 4.1. Comandi per le risorse di sistema

Comando/Sintassi	Cosa fa
`chsh` (`passwd -e/-s`) `username login_shell`	cambia la shell di login dell'utente (spesso solo attraverso il super-user)
`date [opzioni]`	visualizza data e ora corrente
`df [opzioni] [risorsa]`	riporta una sintesi dei blocchi del disco e degli inode liberi e usati
`du [opzioni] [directory o file]`	riporta la quantità di spazio di disco usato
`hostname/uname`	visualizza o setta (solamente super-user) il nome della macchina in uso
`kill [opzioni] [-SEGNALE] [pid#] [%job]`	manda un segnale al processo specificato dal numero di processo id (*`pid#`*) o dal numero di controllo del job (*`%n`*). Il segnale di default termina il processo

Comando/Sintassi	Cosa fa
`man [opzioni] comando`	visualizza la pagina (*man*) del manuale relativa al comando specificato
`passwd [opzioni]`	setta o modifica la propria password
`ps [opzioni]`	mostra lo stato dei processi attivi
`script file`	salva tutto ciò che appare sullo schermo in un file fino a quando viene eseguito il comando `exit`
`stty [opzioni]`	setta o visualizza le opzioni del terminale di controllo
`whereis [opzioni] comando`	riporta le posizioni del file binario, del sorgente e della pagina man relative al comando specificato
`which comando`	riporta il percorso (path) del comando specificato o l'alias di shell in uso
`who` oppure `w`	riporta gli utenti «loggati» e i loro processi in esecuzione

4.1.1. df - riepiloga i blocchi del disco e lo spazio usato

Il comando df(1) è usato per riportare il numero di blocchi del disco e di inode liberi e usati per ogni file system. Il formato dell'output e le valide opzioni sono molto specifiche dal sistema operativo e dalla versione del programma in uso.

Sintassi

```
df [opzioni] [risorsa]
```

Opzioni generali

`-l`	solo il file system locale (SVR4)
`-k`	riporta in kilobyte (SVR4)

Esempi:

```
{Unix prompt 1} df
Filesystem      kbytes  used    avail   capacity  Mounted on

/dev/sd0a       20895   19224   0       102%      /
/dev/sd0h       319055  131293  155857  46%       /usr
/dev/sd1g       637726  348809  225145  61%       /usr/local
/dev/sd1a       240111  165489  50611   77%       /home/guardian
peri:/usr/local/backup
                1952573 976558  780758  56%       /usr/local/backup
peri:/home/peri 726884  391189  263007  60%       /home/peri
peri:/usr/spool/mail
                192383  1081    172064  1%        /var/spool/mail
peri:/acs/peri/2
                723934  521604  129937  80%       /acs/peri/2
```

4.1.2. du - mostra lo spazio del disco usato

Il comando du(1) riporta la quantità di spazio di disco usato per i file o per le directory specificate.

Sintassi

```
du [opzioni] [directory o file]
```

Opzioni generali

`-a`	mostra l'uso del disco per ogni file, non solo per le sotto-directory
`-s`	mostra solo la somma totale
`-k`	riporta in kilobyte (SVR4)

Esempi:

```
{Unix prompt 3} du
 1 ./.elm
 1 ./Mail
 1 ./News
20 ./uc
86 .
```

```
{Unix prompt 4} du -a uc
7 uc/Unixgrep.txt
5 uc/editors.txt
1 uc/.emacs
1 uc/.exrc
4 uc/telnet.ftp
1 uc/uniq.tee.txt
20 uc
```

4.1.3. ps - mostra lo stato dei processi attivi

Il comando ps(1) è usato per mostrare i processi correntemente eseguiti sul sistema. Il formato dell'output e le valide opzioni dipendono molto dal sistema operativo e dalla versione del programma in uso.

Sintassi

```
ps [opzioni]
```

Opzioni generali

BSD	SVR4	
`-a`	`-e`	tutti i processi di tutti gli utenti
`-e`		ambiente di esecuzione
`-g`		processi del gruppo amministrativo come `well`
`-l`	`-l`	formato lungo
`-u`	`-u` *`user`*	relazione specifica di un utente
`-x`	`-e`	anche i processi non eseguiti da terminali
	`-f`	lista completa
`-w`		riporta i primi 132 caratteri per linea

Nota

Poichè il comando ps(1) è molto dipendente dal sistema, si raccomanda di consultare le *pagine man* del proprio sistema per i dettagli delle opzioni e per l'interpretazione dell'output di ps(1).

Esempi:

```
{Unix prompt 5} ps
PID     TT      STAT    TIME    COMMAND
15549   p0      IW      0:00    -tcsh (tcsh)
```

```
15588   p0      IW      0:00    man nice
15594   p0      IW      0:00    sh -c less /tmp/man15588
15595   p0      IW      0:00    less /tmp/man15588
15486   p1      S       0:00    -tcsh (tcsh)
15599   p1      T       0:00    emacs Unixgrep.txt
15600   p1      R       0:00    ps
```

4.1.4. kill - termina un processo

Il comando kill(1) manda un segnale a un processo, generalmente per terminarlo.

Sintassi

```
kill [-SEGNALE] id-processo
```

Opzioni generali

`-l`	visualizza i segnali disponibili per kill

Esempi:

```
{Unix prompt 9} kill -l
HUP INT QUIT ILL TRAP IOT EMT FPE KILL BUS SEGV SYS PIPE ALRM TERM ↺
URG STOP
TSTP CONT CHLD TTIN TTOU IO XCPU XFSZ VTALRM PROF WINCH LOST USR1 ↺
USR2
```

Il segnale `-KILL`, anche specificato con `-9` (a causa della nona posizione assunta nella lista precedente), è il maggior segnale comunemente usato con kill(1). A differenza di altri segnali, questo, una volta captato, non può essere ignorato dal programma.

```
{Unix prompt 10} kill -9 15599
[1] + Killed emacs Unixgrep.txt
```

4.1.5. who - mostra gli utenti attuali

Il comando who(1) riporta chi è correntemente «loggato» nel sistema.

Sintassi

```
who [am i]
```

Esempi:

```
beauty condron> who
wmtell          ttyp1 Apr 21 20:15 (apple.acs.ohio-s)
fbwalk          ttyp2 Apr 21 23:21 (worf.acs.ohio-st)
stwang          ttyp3 Apr 21 23:22 (127.99.25.8)
david           ttyp4 Apr 21 22:27 (slip1-61.acs.ohi)
tgardner        ttyp5 Apr 21 23:07 (picard.acs.ohio-)
awallace        ttyp6 Apr 21 23:00 (ts31-4.homenet.o)
```

```
gtl27           ttyp7 Apr 21 23:24 (data.acs.ohio-st)
ccchang         ttyp8 Apr 21 23:32 (slip3-10.acs.ohi)
condron         ttypc Apr 21 23:38 (lcondron-mac.acs)
dgildman        ttype Apr 21 22:30 (slip3-36.acs.ohi)
fcbetz          ttyq2 Apr 21 21:12 (ts24-10.homenet.)
```

```
beauty condron> who am i
beauty!condron ttypc Apr 21 23:38 (lcondron-mac.acs)
```

4.1.6. whereis - riporta le locazioni del programma

Il comando whereis(1) riporta le locazioni del file sorgente, di quello binario e del file delle pagine man associate al comando.

Sintassi

```
whereis [opzioni] comando
```

Opzioni generali

`-b`	riporta solamente i file binari
`-m`	riporta solamente la sezione manuale
`-s`	riporta solamente i file sorgenti

Esempi:

```
brigadier: condron [69]> whereis Mail
Mail: /usr/ucb/Mail /usr/lib/Mail.help /usr/lib/Mail.rc /usr/man/
man1/Mail.1
```

```
brigadier: condron [70]> whereis -b Mail
Mail: /usr/ucb/Mail /usr/lib/Mail.help /usr/lib/Mail.rc
```

```
brigadier: condron [71]> whereis -m Mail
Mail: /usr/man/man1/Mail.1
```

4.1.7. which - riporta il comando trovato

Il comando which(1) riporta il nome del file che sarà eseguito quando il comando specificato viene invocato. Questo può essere un path name assoluto o il primo alias trovato nel proprio path.

Sintassi

```
which comando
```

Esempi:

```
brigadier: condron [73]> which Mail
```

```
/usr/ucb/Mail
```

4.1.8. hostname/uname - nome della macchina

Il comando hostname(1) (`uname -u` su SysV) riporta il nome host della macchina nella quale l'utente è «loggato», esempio:

```
brigadier: condron [91]> hostname
brigadier
```

uname(1) ha opzioni aggiuntive per visualizzare informazioni circa l'hardware del sistema e la versione del software.

4.1.9. script - memorizza la propria schermata di I/O

Il comando script(1) crea una documentazione della propria sessione di I/O. Usando il comando script(1) si possono catturare tutti i dati trasmessi da e per il proprio terminale visuale fino all'uscita (con `exit`) del programma stesso. Può essere utile durante un processo di debugging, per documentare le azioni che si stanno sperimentando o per avere una copia stampabile per una attenta lettura successiva.

Sintassi

```
script [-a] [file] <...> exit
```

Opzioni generali

`-a`	appende l'output al file

Di default, `typescript` è il nome del file usato dal comando script(1).

Ci si deve ricordare di digitare `exit` per terminare la propria sessione script e chiudere così il file `typescript`.

Esempi:

```
beauty condron> script
Script started, file is typescript
```

```
beauty condron> ps
PID TT STAT TIME COMMAND
23323 p8 S 0:00 -h -i (tcsh)
23327 p8 R 0:00 ps
18706 pa S 0:00 -tcsh (tcsh)
23315 pa T 0:00 emacs
23321 pa S 0:00 script
23322 pa S 0:00 script
3400 pb I 0:00 -tcsh (tcsh)
```

```
beauty condron> kill -9 23315
```

```
beauty condron> date
Mon Apr 22 22:29:44 EDT 1996
```

```
beauty condron> exit
exit
Script done, file is typescript
[1] + Killed emacs
```

```
beauty condron> cat typescript
Script started on Mon Apr 22 22:28:36 1996

beauty condron>ps
PID TT STAT TIME COMMAND
23323 p8 S 0:00 -h -i (tcsh)
23327 p8 R 0:00 ps
18706 pa S 0:00 -tcsh (tcsh)
23315 pa T 0:00 emacs
23321 pa S 0:00 script
23322 pa S 0:00 script
3400 pb I 0:00 -tcsh (tcsh)

beauty condron>kill -9 23315

beauty condron>date
Mon Apr 22 22:29:44 EDT 1996

beauty condron>exit
exit

script done on Mon Apr 22 22:30:02 1996
```

```
beauty condron>
```

4.1.10. date - data e ora corrente

Il comando date(1) mostra la data e l'ora corrente. Un super-user può modificare la data e l'ora.

Sintassi

```
date [opzioni] [+formato]
```

Opzioni generali

`-u`	usa Universal Time (o Greenwich Mean Time)
`+formato`	specifica il formato di output
`%a`	abbreviazione giorni, da Sabato a Domenica
`%h`	abbreviazione mesi, da Gennaio a Dicembre
`%j`	giorno dell'anno, da 001 a 366

`%n`	new-line
`%t`	tab
`%y`	ultime due cifre dell'anno, da 00 a 99
`%D`	formato data MM/DD/YY
`%H`	ora, da 00 a 23
`%M`	minuti, da 00 a 59
`%S`	secondi, da 00 a 59
`%T`	formato ora HH:MM:SS

Esempi:

```
beauty condron> date
Mon Jun 10 09:01:05 EDT 1996
```

```
beauty condron> date -u
Mon Jun 10 13:01:33 GMT 1996
```

```
beauty condron> date +%a%t%D
Mon 06/10/96
```

```
beauty condron> date '+%y:%j'
96:162
```

4.2. Comandi di stampa

Tabella 4.2. Comandi di stampa

Comando/Sintassi	Cosa fa
`lpq` (`lpstat`) `[opzioni]`	mostra lo stato dei job (lavori) di stampa
`lpr` (`lp`) `[opzioni] file`	stampa con una stampante specifica
`lprm` (`cancel`) `[opzioni]`	rimuove un job di stampa dalla coda di stampa
`pr [opzioni] [file]`	filtra il file e lo stampa sul terminale

I comandi di stampa permettono di stampare file sullo standard output (pr(1)) o attraverso una stampante (lp(1) e lpr(1)) fino a filtrare l'output. I comandi di stampa di *BSD* e *SysV* usano diversi nomi e diverse opzioni per produrre lo stesso risultato: lpr(1), lprm(1) e lpq(1) contro lp(1), `cancel` e `lpstat` rispettivamente per BSD e SysV, sottopongono, cancellano e verificano lo stato di un job (lavoro) di stampa.

4.2.1. lp/lpr - sottopone un job di stampa

Il comando lp(1) o lpr(1) sottopone il file specificato o lo standard input al demone di stampa per essere stampato. Ad ogni job viene assegnato un unico id di richiesta che può essere usato in seguito per verificare o cancellare il job mentre è nella coda di stampa.

Sintassi

```
lp [opzioni] filename
```

```
lpr [opzioni] filename
```

Opzioni generali

lp	lpr	funzione
`-n` *`numero`*	*`-#numero`*	numero di copie
`-t` *`titolo`*	*`-Ttitolo`*	titolo del job
`-d` *`destinazione`*	*`-Pstampante`*	nome della stampante
`-c`	(default)	copia il file in coda prima di stamparlo
(default)	`-s`	non copia il file in coda prima di stamparlo
`-o` *`opzioni`*		opzioni addizionali, esempio `nobanner`

I file che iniziano con i simboli `%!` sono considerati file contenenti comandi PostScript.

Esempi:

Per stampare il file `ssh.ps`:

```
% lp ssh.ps
request id is lp-153 (1 file(s))
```

Questo sottopone il job nella coda della stampante di default, `lp`, con l'id di richiesta `lp-153`.

4.2.2. lpstat/lpq - verifica lo stato di un job di stampa

Si può verificare lo stato del proprio job di stampa con il comando `lpstat` o lpq(1).

Sintassi

```
lpstat [opzioni]
```

```
lpq [opzioni] [job#] [username]
```

Opzioni generali

lpstat	lpq	funzione
`-d`	(`lp` di default)	lista dei sistemi destinazione di default
`-s`		riassume lo stato di stampa
`-t`		stampa tutte le informazioni dello stato di stampa
`-u [`*`login-ID-list`*`]`		lista dell'utente specificato
`-p` *`stampante`*	`-P`*`stampante`*	stampa lo stato della stampante specificata

Esempi:

```
% lpstat
lp-153 frank 208068 Apr 29 15:14 on lp
```

4.2.3. cancel/lprm - cancella un job di stampa

Alcuni utenti possono cancellare solamente i loro job di stampa.

Sintassi

```
cancel [id-richiesta] [stampante]
```

```
lprm [opzioni] [job#] [username]
```

Opzioni generali

cancel	lprm	funzione
	`-P`*`stampante`*	specifica la stampante
	-	tutti i job dell'utente
`-u [`*`login-ID-list`*`]`		lista dell'utente

Esempi:

Per cancellare il job sottomesso in precedenza:

```
% cancel lp-153
```

4.2.4. pr - prepara file per la stampa

Il comando pr(1) stampa l'intestazione e le informazioni traccia che circoscrivono il file formattato. Si può specificare il numero di pagine da stampare, le linee per pagina, le colonne, le linee bianche, si può specificare la larghezza di pagina, l'intestazione e le informazioni traccia e in che modo trattare il carattere tab.

Sintassi

```
pr [opzioni] file
```

Opzioni generali

+numero_pagina	inizia a stampare al numero di pagina specificato
-colonne	numero di colonne
-a	modifica l'opzione *-colonne* per riempire le colonne nell'ordine round-robin
-d	doppio spazio
-e [*carattere*] [*gap*]	spazio tab
-h *stringa_intestazione*	intestazione per ogni pagina
-l *linee*	linee per pagina
-t	non stampa l'intestazione e la traccia per ogni pagina
-w *larghezza*	larghezza di pagina

Esempi:

Il file contenente la lista di libri di P. G. Wodehouse Lord Emsworth dovrebbe essere stampato con 14 linee per pagina (incluse 5 di intestazione e 5 (vuote) linee traccia), dove l'opzione -e specifica in che modo convertire i tab:

```
% pr -l 14 -e42 wodehouse




Apr 29 11:11 1996 wodehouse_emsworth_books Page 1



Something Fresh [1915]                  Uncle Dynamite [1948]
Leave it to Psmith [1923]               Pigs Have Wings [1952]
Summer Lightning [1929]                 Cocktail Time [1958]
Heavy Weather [1933]                    Service with a Smile [1961]




Apr 29 11:11 1996 wodehouse_emsworth_books Page 2
```

```
Blandings Castle and Elsewhere [1935]   Galahad at Blandings [1965]
Uncle Fred in the Springtime [1939]     A Pelican at Blandings [1969]
Full Moon [1947]                        Sunset at Blandings [1977]
```

Capitolo 5. Shell

La shell, che sta tra l'utente e il sistema operativo, opera come un interprete di comandi. Legge l'input dal terminale e traduce i comandi in azioni, azioni che vengono intraprese dal sistema. La shell è simile al *command.com* in DOS. Una volta effettuato il login nel sistema, viene assegnata la shell di default. La shell, al suo avvio, legge i suoi file di inizializzazione e può settare alcune variabili di ambiente, i path di ricerca dei comandi, gli alias dei comandi ed eseguire qualche comando specificato in questi file.

La prima shell è stata la shell Bourne, sh(1). Ogni piattaforma Unix dispone della shell Bourne o di una shell Bourne compatibile. Questa shell ha molte buone caratteristiche per controllare l'input e l'output, ma non è molto adatta all'utente interattivo. Per andare incontro a quest'ultimo è stata scritta la shell C, csh(1), presente ora in molti, ma non tutti, i sistemi Unix. Questa shell usa una sorta di sintassi C, il linguaggio con cui Unix è stato scritto, ma ha molte scomode implementazioni dell'input/output. La shell C ha il controllo dei job, quindi può mandare un job eseguito in background («sotto shell») in foreground («in shell corrente»). Inoltre ha la funzione di history (storia dei comandi) che permette di modificare e ripetere comandi eseguiti precedentemente.

Il prompt di default per la shell Bourne è *$* (o *#* per l'utente root). Il prompt di default per la shell C è *%*.

Sono disponibili in rete molte altre shell. Quasi tutte sono basate sulla shell sh(1) o csh(1) con estensioni per fornire il controllo dei job alla shell sh(1), permettere di manipolare il modo di esecuzione dei comandi su linea di comando, cercare attraverso i comandi eseguiti precedentemente, fornire il completamento dei nomi dei comandi, prompt personalizzati, ecc. Alcune delle seguenti shell maggiormente conosciute potrebbero essere sul proprio amato sistema Unix: la shell korn, `ksh`, di David Korn e la shell Bourne Again, bash(1), dal progetto GNU Free Software Foundations, entrambe basate su sh(1), la shell T-C, tcsh(1) e l'estensione della shell C, `cshe`, entrambe bastate su csh(1). In seguito si descriveranno alcune delle caratteristiche di sh(1) e csh(1), così per iniziare.

5.1. Comandi built-in

La shell ha alcuni comandi *built-in*, chiamati anche comandi nativi. Questi comandi sono eseguiti direttamente dalla shell e non chiamano nessun altro programma per essere eseguiti. Questi comandi built-in possono essere diversi tra le varie shell.

5.1.1. Sh

Per la shell Bourne alcuni dei comandi built-in più comunemente usati sono:

:	comando null

`.`	prende (legge ed esegue) i comandi da un file
`case`	condizionale case
`cd`	cambia la directory di lavoro (`$HOME` di default)
`echo`	scrive una stringa su standard output
`eval`	valuta l'argomento specificato e ritorna il risultato alla shell
`exec`	esegue il comando specificato rimpiazzando la shell corrente
`exit`	esce dalla shell corrente
`export`	condivide le variabili di ambiente specificate con le successive shell
`for`	condizionale di ciclo for
`if`	condizionale if
`pwd`	mostra la directory di lavoro corrente
`read`	legge una linea di input da standard input
`set`	setta le variabili di shell
`test`	valuta un'espressione come vera o falsa
`trap`	intrappola un tipo di segnale ed esegue comandi
`umask`	setta la maschera di default relativa ai permessi da impostare per i nuovi file
`unset`	resetta le variabili di shell
`wait`	attente che un specifico processo termini
`while`	condizionale di ciclo while

5.1.2. Csh

Per la shell C i comandi built-in maggiormente usati sono:

`alias`	assegna un nome a una funzione
`bg`	mette un job in background
`cd`	cambia la directory di lavoro corrente
`echo`	scrive una stringa su standard output
`eval`	valuta gli argomenti specificati e ritorna il risultato alla shell
`exec`	esegue il comando specificato rimpiazzando la shell corrente
`exit`	esce dalla shell corrente
`fg`	porta un job in foreground
`foreach`	condizionale di ciclo for

`glob`	crea un'espansione dei nomi di file su una lista senza tenere conto del carattere di escape \
`history`	stampa la storia dei comandi della shell
`if`	condizionale if
`jobs`	mostra o controlla i job attivi
`kill`	termina un processo specifico
`limit`	setta dei limiti sulle risorse di sistema
`logout`	termina la shell di login
`nice comando`	abbassa la priorità di schedulazione del comando specificato
`nohup comando`	non termina il comando specificato quando la shell esce
`popd`	estrae un record dallo stack delle directory e ritorna nella directory estratta
`pushd`	cambia nella nuova directory specificata e aggiunge quella corrente nello stack delle directory
`rehash`	ricrea la tabella hash dei percorsi (path) per i file eseguibili
`repeat`	ripete un comando il numero di volte specificato
`set`	setta una variabile di shell
`setenv`	setta una variabile di ambiente per la shell corrente e per quelle successive
`source`	prende (legge ed esegue) comandi da un file
`stop`	ferma uno specifico job in background
`switch`	condizionale switch
`umask`	setta la maschera di default relativa ai permessi da impostare per i nuovi file
`unalias`	rimuove il nome alias specificato
`unset`	resetta le variabile di shell
`unsetenv`	resetta le variabili di ambiente
`wait`	attente la terminazione di tutti i processi in background
`while`	condizionale di ciclo while

5.2. Variabili di ambiente

Le variabili di ambiente sono usate per fornire informazioni ai programmi che si utilizzano. Si possono avere sia *variabili globali di ambiente* sia *variabili locali di shell*. Le variabili glo-

bali di ambiente sono inizializzate attraverso la propria shell di login e i nuovi programmi e le nuove shell ereditano l'ambiente della shell genitore. Le variabili locali di shell sono usate solamente dalla shell corrente e non sono passate ad altri processi. Un processo figlio non può passare una variabile al suo processo padre.

Le variabili di ambiente correnti sono visualizzabili con i comandi env(1) o printenv(1). Alcune comuni variabili sono:

DISPLAY	Il display grafico da usare, esempio `nyssa:0.0`
EDITOR	Il path (percorso) del proprio editor di default, esempio `/usr/bin/vi`
GROUP	Il proprio gruppo di login, esempio `staff`
HOME	Il path della propria home directory, esempio `/home/frank`
HOST	Il nome host del proprio sistema, esempio `nyssa`
IFS	I separatori di campo interni, generalmente alcuni spazi bianchi (tab, spazio e new-line di default)
LOGNAME	Il nome del proprio login, esempio `frank`
PATH	I path per ricercare i comandi, esempio `/usr/bin:/usr/ucb:/usr/local/bin`
PS1	La stringa del prompt primario, solamente shell Bourne ($ di default)
PS2	La stringa del prompt secondario, solamente shell Bourne (> di default)
SHELL	La propria shell di login, esempio `/usr/bin/csh`
TERM	Il proprio tipo di terminale, esempio `xterm`
USER	Il proprio username, esempio `frank`

Molte variabili di ambiente sono inizializzate automaticamente quando si effettua il login. Queste possono essere modificate e si possono definire altre variabili nei propri file di inizializzazione o in qualunque momento all'interno della shell. Alcune variabili che è possibile si voglia cambiare sono PATH e DISPLAY. La variabile PATH specifica le directory nelle quali saranno automaticamente cercati i comandi richiesti. Alcuni esempi sono nello script di inizializzazione di shell mostrato più avanti.

Per la shell C si può settare una *variabile globale di ambiente* con un comando simile a quello usato per visualizzare le variabili:

```
% setenv NOME valore
```

e per la shell Bourne:

```
$ NOME=valore; export NOME
```

Si possono elencare le proprie variabili globali di ambiente con i comandi env(1) o printenv(1). Si possono resettare queste variabili con i comandi `unsetenv` (shell C) o `unset` (shell Bourne).

Per settare una *variabile locale di shell* in shell C si usa il comando `set` con la sintassi seguente. Senza opzioni `set` mostra tutte le variabili locali.

```
% set nome=valore
```

Per la shell Bourne si setta una variabile locale di shell con:

```
$ nome=valore
```

Il valore corrente di una variabile è accessibile attraverso le notazioni `$nome` o `${nome}`.

5.3. La shell Bourne, sh

La shell sh(1) usa il file di inizializzazione `.profile` posto nella home directory dell'utente. Inoltre può esserci un file di inizializzazione globale del sistema, esempio `/etc/profile`. In tal caso, il file globale del sistema sarà eseguito prima di quello locale.

Un semplice file `.profile` potrebbe essere come il seguente:

```
PATH=/usr/bin:/usr/ucb:/usr/local/bin:.         # setta il PATH
export PATH                   # rende disponibile PATH per le sotto-
shell
# setta il prompt
PS1="{`hostname` `whoami`} "     # setta il prompt, $ di default
# funzioni
ls() { /bin/ls -sbF "$@";}
ll() { ls -al "$@";}
# setta il tipo di terminale
stty erase ^H              # setta Control-H come tasto di ↺
cancellazione
eval `tset -Q -s -m ':?xterm'` # richiede il tipo di terminale, ↺
presupponendo
                               # xterm
#
umask 077
```

Ogni volta che si incontra il simbolo `#`, il resto di quella linea viene trattato come un commento. Nella variabile `PATH` ogni directory è separata da due punti (:) e il punto (`.`) specifica che la directory corrente è nel proprio path. Se il punto non è nel proprio path, lo stesso diventa un semplice elemento per eseguire un programma nella directory corrente:

```
./programma
```

Non è una buona idea avere il punto (`.`) nel proprio `PATH`, in modo tale da non eseguire inavvertitamente un programma senza averne l'intenzione quando si usa il comando `cd` per spostarsi in differenti directory.

Una variabile settata in `.profile` rimane valida solo nel contesto della shell di login, a meno che la si esporti con `export` o si esegua `.profile` in un'altra shell. Nell'esempio precedente `PATH` viene esportato per le sotto-shell. Si può eseguire un file con il comando built-int `.` di sh(1), esempio:

```
. ./.profile
```

Si possono creare proprie funzioni. Nell'esempio precedente la funzione `ll`, risultato di `ls -al`, lavora su un specifico file o directory.

Con stty(1) il carattere di cancellazione viene settato a Control+H, che è usualmente il tasto di Backspace.

Il comando tset(1) richiede il tipo di terminale e assume questo a `xterm` se si conferma con invio <CR>. Questo comando è eseguito con un comando built-in di shell, `eval`, che prende il risultato del comando tset(1) e lo usa come argomento per la shell. In questo caso l'opzione `-s` di tset(1) setta le variabili `TERM` e `TERMCAP` e le esporta.

L'ultima linea nell'esempio richiama il comando `umask`, facendo in modo che i file e le directory create non abbiano i permessi di lettura-scrittura-esecuzione per l'utenza *gruppo* e *altri*.

Per altre informazioni su sh(1), digitare `man sh` al prompt di shell.

5.4. La shell C, csh

La shell C, csh(1), usa i file di inizializzazione `.cshrc` e `.login`. Alcune versioni usano un file di inizializzazione globale del sistema, ad esempio `/etc/csh.login` . Il proprio file `.login` è eseguito solamente quando si effettua il login. Il proprio file `.cshrc` è eseguito ogni volta in cui si avvia una shell csh(1), incluso quando si effettua il login. Questo file ha molte caratteristiche simili al file `.profile`, ma un differente modo di composizione. Qui si usano i comandi `set` o `setenv` per inizializzare una variabile, dove `set` viene usato per definire una variabile solo per la shell corrente, mentre `setenv` definisce una variabile per la shell corrente e per le altre sotto-shell. Le variabili di ambiente `USER`, `TERM` e `PATH` sono automaticamente importate ed esportate dalle variabili `user`, `term` e `path` della shell csh(1). Quindi `setenv` non è necessario per queste variabili. La shell C usa il simbolo ~ per indicare la directory home dell'utente in un path, come in `~/.cshrc` o per specificare una directory di login di un altro utente, come in `~username/.cshrc` .

Alcune variabili predefinite usate dalla shell C sono:

`argv`	La lista degli argomenti della shell corrente
`cwd`	La directory di lavoro corrente
`history`	Imposta la dimensione della lista di history (storia) da memorizzare
`home`	La directory home dell'utente, visualizzabile con `$HOME`
`ignoreeof`	Quando viene settata, EOF (Ctrl+D) viene ignorato dal terminale
`noclobber`	Quando viene settata si impedisce di redirigere l'output per sovrascrivere file esistenti
`noglob`	Quando viene settata si impedisce l'espansione dei nomi di file all'interno di un confronto con uno schema wild card
`path`	I path di ricerca dei comandi, visualizzabile con `$PATH`
`prompt`	Setta il prompt della linea di comando (`%` di default)
`savehist`	Numero di volte (di login) che bisogna mantenere memorizzata la lista di history nel file `.history`
`shell`	Il path name completo della shell corrente, visualizzabile con `$SHELL`
`status`	Il codice di stato di uscita dell'ultimo comando (`0`=uscita normale, `1`=comando fallito)
`term`	Il proprio tipo di terminale, visualizzabile con `$TERM`
`user`	Il proprio nome utente, username, visualizzabile con `$USER`

Un semplice file `.cshrc` potrebbe essere come il seguente:

```
set path=(/usr/bin /usr/ucb /usr/local/bin ~/bin . ) # setta il path
set prompt = "{`hostname` `whoami` !} "   # setta il promt ↺
primario -;
                                          # % di default
set noclobber                  # non redirige l'output su file ↺
esistenti
set ignoreeof                      #ignora EOF (Ctrl+D) in questa ↺
shell
set history=100 savehist=50   # mantiene una lista history di ↺
comandi e la
                              # memorizza tra vari (50) login
# alias
alias h history                    # alias h per history
alias ls "/usr/bin/ls -sbF"        # alias ls per ls -sbF
alias ll ls -al              # alias ll per ls -sbFal (combina queste
                             # opzioni con quelle di ls sopra citate)
alias cd 'cd \!*;pwd'        # alias cd per stampare la directory ↺
di lavoro
```

```
                                # corrente dopo aver cambiato directory
umask 077
```

Alcune nuove caratteristiche che non sono state viste nel file `.profile` (shell sh(1)) sono `noclobber`, `ignoreeof` e `history`. `Noclobber` indica che l'output non può essere rediretto su un file esistente, mentre `ignoreeof` specifica che EOF (Ctrl+D) non causa l'uscita dalla shell di login o l'uscita dal sistema.

Con la caratteristica di *history* si possono richiamare comandi eseguiti precedentemente e rieseguirli, eventualmente con dei cambiamenti.

Un *alias* permette di usare uno specifico nome *alias* al posto del comando completo. Nell'esempio precedente, il risultato di digitare `ls` sarà quello di eseguire `/usr/bin/ls -sbF`. Si può verificare quale comando `ls` è nel proprio path con il comando which(1), ad esempio:

```
% which ls
ls:     alias di /usr/bin/ls -sbF
```

Un semplice file `.login` potrebbe essere come il seguente:

```
# .login
stty erase ^H      # setta Control+H come tasto di eliminazione
set noglob     # impedisce un confronto con uno schema wild card
eval `tset -Q -s -m ':?xterm'`  # chiede il tipo di terminale ↺
presupponendo
                                # xterm
unset noglob       # riabilita un confronto con uno schema wild card
```

Abilitando e disabilitando `noglob` intorno a tset(1) si impedisce di confondere il tipo di terminale con qualche espansione dei nomi di file in un confronto con uno schema (pattern) wild card.

Se si effettuano cambiamenti al proprio file di inizializzazione, questi possono essere attivati eseguendo il file modificato. Per la shell csh(1) questo è possibile attraverso il comando built-in `source`, esempio:

```
source .cshrc
```

Per altre informazioni circa la shell csh(1) digitare `man csh` al prompt di shell.

5.5. Controllo dei job

Con la shell C, csh(1) e molte altre nuove shell, incluse alcune nuove shell Bourne, si possono mettere i job in background apporrendo *&* al comando, così come succede per la shell sh(1). Questo può anche essere fatto, una volta sottoposto il comando, digitando Control+Z per sospendere il job e quindi `bg` per metterlo in background. Per riportarlo in foreground si digita `fg`.

Si possono avere molti job eseguiti in background. Quando questi sono in background, non sono connessi alla tastiera per l'input, ma possono tuttavia mostrare l'output nel terminale, sparpagliandolo con qualsiasi cosa ci sia digitata o mostrata attraverso il job corrente. Si può avere la necessità di redirigere I/O in o da un file per un job in background. La propria tastiera è solamente connessa al corrente job in foreground.

Il comando built-in `jobs` permette di elencare i propri job in background. Si può usare il comando kill(1) per terminare un job in background. In questi comandi, con la notazione *`%n`* ci si riferisce all'*`n-esimo`* job in background, rimpiazzando *`n`* con il numero di job proveniente dall'output di `jobs`. Quindi si termina il secondo job in background con `kill %2` e si riprende il terzo job in foreground con `fg %3`.

5.6. History

La shell C, la shell Korn e molte altre shell avanzate, mantengono informazioni sui comandi che sono stati eseguiti in shell. La quantità di storia memorizzabile dipende dalla shell utilizzata. Qui si descriveranno le caratteristiche di history della shell C.

Si possono usare le variabili `history` e `savehist` per settare rispettivamente quanti comandi della shell corrente memorizzare e per quanti login mantenerli. Si può inserire in `.cshrc` la seguente linea per memorizzare 100 comandi della shell corrente fino a 50 prossimi login.

```
set history=100 savehist=50
```

La shell mantiene traccia della storia dei comandi tra un login e l'altro memorizzandola nel file `~/.history`.

Si può usare il comando built-in `history` per richiamare i comandi eseguiti precedentemente, ad esempio per stampare gli ultimi 10:

```
% history 10
52 cd workshop
53 ls
54 cd Unix_intro
55 ls
56 pwd
57 date
58 w
59 alias
60 history
61 history 10
```

Si può ripetere l'ultimo comando digitando `!!`:

```
% !!
53 ls
54 cd Unix_intro
```

```
55 ls
56 pwd
57 date
58 w
59 alias
60 history
61 history 10
62 history 10
```

Si può ripetere un comando numerato introducendo il numero con un `!`, esempio:

```
% !57
date
Tue Apr 9 09:55:31 EDT 1996
```

Si può ripetere un comando che inizia con qualche stringa, introducendo la parte iniziale univoca della stringa con un `!`, esempio:

```
% !da
date
Tue Apr 9 09:55:31 EDT 1996
```

Quando la shell valuta la linea di comando verifica subito la sostituzione di history prima di interpretare qualche altra cosa. Per usare uno di questi caratteri speciali in un comando di shell è necessario usare un escape, o effettuare un quoting, apporrendo un \ prima del carattere, esempio \!. I caratteri di sostituzione di history sono sintetizzati nella tabella seguente:

Tabella 5.1. Comandi di sostituzione di history per la shell C

Comando	Funzione sostitutiva
`!!`	ripete l'ultimo comando
`!n`	ripete il comando numero *n*
`!-n`	ripete l'*n-esimo* comando partendo dall'ultimo
`!str`	ripete il comando che inizia con la stringa *str*
`!?str?`	ripete il comando con all'interno *str*
`!?str?%`	seleziona il primo argomento che ha *str* all'interno
`!:`	ripete l'ultimo comando, generalmente usato con una modifica
`!:n`	seleziona l'*n-esimo* argomento dell'ultimo comando (n=0 è il nome del comando)
`!:n-m`	seleziona gli argomenti tra l'*n-esimo* e l'*m-esimo* argomento dell'ultimo comando
`!^`	seleziona il primo argomento dell'ultimo comando (come `!:1`)

Comando	Funzione sostitutiva
`!$`	seleziona l'ultimo argomento dell'ultimo comando
`!*`	seleziona tutti gli argomenti del precedente comando
`!:n*`	seleziona gli argomenti dall'*`n-esimo`* all'ultimo, incluso, del precedente comando
`!:n-`	seleziona gli argomenti dall'*`n-esimo`* all'ultimo, escluso, del precedente comando
`^str1^str2^`	rimpiazza *`str1`* con *`str2`* nella prima occorrenza nel precedente comando
`!n:s/str1/str2/`	sostituisce *`str1`* con *`str2`* nella prima occorrenza nell'*`n-esimo`* comando, finendo con una sostituzione globale *g*

Altre informazioni sono descritte nelle *pagine man*.

5.7. Cambiare la propria shell

Per cambiare la propria shell si usano generalmente i comandi chsh(1) o `passwd -e`. Il flag di opzione, qui `-e`, può cambiare da sistema a sistema (`-s` su sistemi basati su BSD), quindi verificare le proprie *pagine man* sul proprio sistema per un uso corretto. Alcune volte questa caratteristica è disabilitata. Se non si riesce a cambiare la propria shell contattare il proprio amministratore di sistema (System Administrator).

La nuova shell deve essere un path name assoluto di una valida shell sul sistema. Le shell disponibili variano da sistema a sistema. Inoltre il path name assoluto della shell può cambiare. Normalmente, per la shell Bourne e la shell C sono standard e sono:

```
/bin/sh
```

```
/bin/csh
```

Alcuni sistemi hanno anche la shell Korn standard normalmente in:

```
/bin/ksh
```

Altre shell, che sono poco popolari e non distribuite normalmente dal venditore di OS, sono bash(1) e tcsh(1). Queste potrebbero essere situate in `/bin` o in una directory locale, esempio `/usr/local/bin` o `/opt/local/bin` . Se si sceglie una shell non standard del OS, ci si deve assicurare che quella shell e tutte le shell di login disponibili sul sistema siano elencate nel file `/etc/shells` . Se questo file esiste e la propria shell non è elencata in esso, il demone per il trasferimento di file, ftpd(8), ti impedirà una connessione ftp su questa macchina. Se tale file non esiste, solamente account con shell «standard» possono connettersi via ftp(1).

Si può sempre provare una shell prima di settarla come la propria shell di default. Per fare questo si deve digitare il nome della shell che si desidera utilizzare, come qualsiasi altro comando.

Capitolo 6. Caratteristiche speciali di Unix

Uno dei più importanti contributi che Unix ha dato ai sistemi operativi è stato quello di fornire molti strumenti per creare lavori ordinari e per ottenere le informazioni che si desiderano. Un altro è rappresentato dal modo standard con cui i dati sono memorizzati e trasmessi in un sistema Unix. Questo permette di trasferire dati *in* un file, nel terminale video o in un programma, oppure *da* un file, dalla tastiera o da un programma, sempre in maniera uniforme. Il trattamento standardizzato dei dati supporta due importanti caratteristiche di Unix: la redirezione di I/O e il piping.

Con la *redirezione dell'output*, l'output di un comando viene rediretto su un file piuttosto che sul terminale video. Con la *redirezione dell'input*, l'input di un comando viene preso da un file piuttosto che dalla tastiera. Sono possibili altre tecniche di redirezione dell'input e dell'output come si vedrà in seguito. Con il *piping*, l'output di un comando può essere usato come input di un comando successivo. In questo capitolo si discuterà di alcune delle caratteristiche e degli strumenti disponibili per gli utenti Unix.

6.1. Descrittori di file

Ci sono 3 descrittori di file standard:

stdin(4)	0	Standard input per il programma
stdout(4)	1	Standard output dal programma
stderr(4)	2	Standard error (output) dal programma

Normalmente l'input viene preso dalla tastiera o da un file. Generalmente l'output, sia stdout(4) che stderr(4), scorre nel terminale, ma può essere rediretto, uno o entrambi, su uno o più file.

Si possono specificare descrittori di file addizionali, denotandoli con un numero da 3 a 9 e redirigendo l'I/O attraverso questi.

6.2. Redirezione di file

La redirezione dell'output prende l'output di un comando e lo posiziona nel file specificato. La redirezione dell'input legge il file specificato come input per un comando. La tabella che segue sintetizza le possibili modalità di redirezione.

Tabella 6.1. Redirezione di file

SIMBOLO	REDIREZIONE
`>`	redirezione dell'output
`>!`	come sopra, ma non tiene conto dell'opzione *noclobber* per csh(1)
`>>`	appende l'output
`>>!`	come sopra, ma non tiene conto dell'opzione *noclobber* su csh(1) e crea il file se non esiste
`|`	incanala (pipe) l'output nell'input di un altro comando
`<`	redirezione dell'input
`<<Stringa`	legge da standard input fino a quando incontra una linea contenente solo la parola *`Stringa`*. Anche conosciuto come *here document* (vedere il Capitolo 9)
`<<\Stringa`	come sopra, ma le sostituzioni di shell non sono permesse

Un esempio di redirezione dell'output è:

```
cat file1 file2 > file3
```

Il precedente comando concatena `file1` e `file2` e redirige (manda) l'output in `file3`. Se `file3` non esiste, viene creato. Se esiste, verrà troncato a lunghezza zero prima che il nuovo contenuto sia inserito, oppure, se l'opzione *noclobber* della shell csh(1) è abilitata, il comando verrà rifiutato (vedere la shell csh(1) nel Capitolo 5). I file originali `file1` e `file2` rimarranno come erano prima dell'esecuzione del comando, ossia due entità separate.

L'output viene appeso a un file con la forma:

```
cat file1 >> file2
```

Questo comando appende il contenuto di `file1` alla fine dell'esistente `file2` (`file2` non viene soprascritto).

L'input è rediretto (preso) da un file con la forma:

```
programma < file
```

Questo comando prende l'input per il `programma` da `file`.

Per incanalare (pipe) l'output di un programma nell'input di un altro programma si usa la forma:

```
comando|comando
```

Questo comando assegna l'output del primo comando all'input del secondo comando.

6.2.1. Csh

`>& file`	redirige stdout(4) e stderr(4) in `file`
`>>& file`	appende stdout(4) e stderr(4) in `file`
`|& comando`	crea una pipe tra stdout(4)-stderr(4) e il `comando`

Per redirigere stdout(4) e stderr(4) in due file separati si deve prima redirigere stdout(4) in una sotto-shell, così:

```
% (comando > out_file) >& err_file
```

6.2.2. Sh

`2> file`	redirige stderr(4) in `file`
`> file 2>&1`	redirige stdout(4) e stderr(4) in `file`
`>> file 2>&1`	appende stdout(4) e stderr(4) in `file`
`2>&1 | comando`	crea una pipe tra stdout(4)-stderr(4) e il *comando*

Per redirigere stdout(4) e stderr(4) in due file separati si può fare:

```
$ comando 1> out_file 2> err_file
```

oppure, data la redirezione di default per stdout(4):

```
$ comando > out_file 2> err_file
```

Con la shell Bourne si possono specificare altri descrittori di file (da 3 a 9) e redirigere l'output attraverso questi. Questo può essere fatto con la forma:

`n>&m`	redirige il descrittore di file *n* sul descrittore di file *m*

Questo meccanismo viene utilizzato per mandare stderr(4) nello stesso posto di stdout(4), `2>&1`, quando si vuole avere i messaggi di errore e i normali messaggi in un file piuttosto che sul terminale. Se si vuole che solamente i messaggi di errore vadano nel file, si può usare un descrittore di file di supporto, 3. Si redirige prima 3 su 2, quindi 2 su 1 e in fine si redirige 1 su 3.

```
$ (comando 3>&2 2>&1 1>&3) > file
```

Questo manda stderr(4) in 1 e stdout(4) in 3 che è rediretto su 2. In questo modo, in effetti, si ribaltano i normali significati dei descrittori di file 1 e 2. Si può sperimentare tutto questo con l'esempio seguente:

```
$ (cat file 3>&2 2>&1 1>&3) > errfile
```

Quindi se `file` è letto, l'informazione è scartata dall'output del comando, ma se `file` non può essere letto, i messaggi di errore sono messi nel file `errfile` per usi futuri.

I descrittori di file che sono stati creati possono essere chiusi con:

`m<&-`	chiude un descrittore di file di input
`<&-`	chiude stdin(4)
`m>&-`	chiude un descrittore di file di output
`>&-`	chiude stdout(4)

6.3. Altri speciali simboli di comando

Oltre ai simboli di redirezione dei file ci sono altri simboli speciali che si possono usare su linea di comando. Alcuni di questi sono:

`;`	separatore di comandi
`&`	esegue un comando in background
`&&`	esegue il comando seguente (a questo simbolo) solamente se il comando precedente (a questo simbolo) è stato completato con successo, esempio: `grep stringa file && cat file`
`||`	esegue il comando seguente (a questo simbolo) solamente se il comando precedente (a questo simbolo) non è stato completato con successo, esempio: `grep stringa file || echo "Stringa non trovata."`
`( )`	i comandi tra parentesi sono eseguiti in una sotto-shell. L'output della sotto-shell può essere manipolato come specificato nelle precedenti sezioni.
`' '`	segni di quoting letterali. All'interno di questi segni di quoting non viene permesso ad alcuni caratteri di assumere significati speciali.
`\`	considera il prossimo carattere letteralmente (escape)
`" "`	segni di quoting regolari. All'interno di questi segni di quoting sono permesse sostituzioni di variabili e di comando (non disattivano $ e \ all'interno della stringa).
`` `comando` ``	prende l'output del comando e lo sostituisce nell'argomento su linea di comando
`#`	ogni cosa che lo segue fino a un newline è un commento

Inoltre, il carattere \ può essere usato per effettuare un escape sul carattere di newline, in modo tale da continuare un lungo comando su più di una linea fisica di testo.

6.4. Meta caratteri

La shell e alcuni programmi di manipolazione testo permettono i *meta-caratteri*, chiamati anche *wild card*, i quali vengono rimpiazzati dai corrispondenti schemi (pattern). Per i nomi di file questi *meta-caratteri* e i loro significati sono:

?	indica un singolo carattere alla posizione indicata
***	indica una stringa di zero o più caratteri
[abc...]	indica un carattere tra quelli racchiusi
[a-e]	indica un carattere tra quelli nel range a, b, c, d, e
[!def]	indica un carattere tra quelli non inclusi in parentesi, solamente sh(1)
{abc,bcd,cde}	indica un set di caratteri tra quelli inclusi in parentesi, separati da una virgola (,) (niente spazi), solamente csh(1)
~	indica la directory home dell'utente corrente, solamente csh(1)
~user	indica la directory home dell'utente specificato, solamente csh(1)

Capitolo 7. Manipolazione del testo

7.1. Sintassi delle espressioni regolari

Alcuni programmi di manipolazione del testo come grep(1), egrep(1), sed(1), awk(1) e vi(1) consentono di ricercare uno schema (pattern) piuttosto che una stringa fissa. Questi schemi testuali sono conosciuti come *espressioni regolari*. Si può formare un'espressione regolare combinando caratteri normali con caratteri speciali, anche conosciuti come *meta-caratteri*, secondo le successive regole. Con queste espressioni regolari si può *confrontare uno schema* su dati testuali. Le espressioni regolari si presentano in tre diverse forme:

Ancoraggi	legano lo schema a una posizione sulla linea
Serie di caratteri	indicano un carattere in una singola posizione
Modificatori	specificano quante volte ripetere l'espressione precedente

Segue la sintassi delle espressioni regolari. Alcuni programmi accettano tutte queste sintassi, altri ne accettano solo alcune:

.	indica *un* singolo carattere eccetto quello di newline
*	indica *zero o più* istanze del singolo carattere (o meta-carattere) che lo precede
[abc]	indica un carattere tra quelli racchiusi
[a-d]	indica un carattere tra quelli compresi nel range
[^exp]	indica un carattere tra quelli *non* inclusi nell'espressione
^abc	l'espressione regolare deve iniziare all'*inizio della linea* (Ancoraggio)
abc$	l'espressione regolare deve finire alla *fine della linea* (Ancoraggio)
\	tratta il carattere successivo letteralmente. Viene normalmente usato per mantenere inalterato il significato di un carattere speciale come . e *.
\{n,m\}	confronta l'espressione regolare precedente un numero minimo *n* di volte e un numero massimo *m* di volte (*n* e *m* possono assumere valori tra 0 e 255). I simboli \{ e \} dovrebbero essere intesi come singoli operatori. In questo caso il simbolo \ che precede le parentesi non è il carattere di escape, ma assume un nuovo significato.
\<abc\>	confronta l'espressione regolare racchiusa trattandola come una singola parola. I limiti della parola sono definiti iniziando con un newli-

	ne o qualche altra cosa, eccetto una lettera, una cifra o un underscore (_), e finendo con la stessa cosa o con un carattere di fine linea. Ancora, i simboli \< e \> dovrebbero essere intesi come singoli operatori.
\(*abc*\)	salva lo schema racchiuso in un buffer. Possono essere salvati per ogni linea fino a nove schemi. È possibile riferirsi a questi schemi tramite la combinazione di caratteri *n*. Ancora una volta i simboli \(e \) dovrebbero essere intesi come singoli operatori.
n	dove *n* varia tra 1 e 9. Confronta l'*n-sima* espressione precedentemente salvata per la linea corrente. Le espressioni sono numerate partendo da sinistra. Il simbolo *n* dovrebbe essere inteso come un singolo operatore.
&	mostra lo schema di ricerca precedente (usato al posto della stringa)

Ci sono alcuni meta-caratteri usati solamente da awk(1) e egrep(1). Questi sono:

+	confronta una o più delle espressioni precedenti (a questo simbolo)
?	confronta zero o alcune delle espressioni precedenti (a questo simbolo)
|	separatore. Confronta sia l'espressione precedente (a questo simbolo) sia quella seguente
( )	raggruppa le espressioni regolari all'interno delle parentesi e applica una serie di confronti

Alcuni esempi di *espressioni regolari* comuni sono:

espressione regolare	indica
cat	la stringa *cat*
.at	alcune occorrenze di un carattere precedente ad *at*, come cat, rat, mat, bat, fat, hat
xy*z	alcune occorrenze di un *x*, seguite da zero o più *y* e seguite da una *z*.
^cat	*cat* all'inizio della linea
cat$	*cat* alla fine della linea
*	alcune occorrenze di un asterisco
[cC]at	*cat* o *Cat*
[^a-zA-Z]	alcune occorrenze di caratteri non alfabetici
[0-9]$	alcune linee che finiscono con un numero
[A-Z][A-Z]*	una o più lettere maiuscole
[A-Z]*	zero o alcune lettere maiuscole (in altre parole, qualcosa)

7.2. Comandi di manipolazione del testo

Tabella 7.1. Comandi di manipolazione del testo

Comando/Sintassi	Cosa fa
`awk/nawk [opzioni] file`	esamina gli schemi (pattern) all'interno di un file ed elabora i risultati
`grep/egrep/fgrep [opzioni] 'stringa di ricerca' file`	ricerca nell'argomento (in questo caso probabilmente un file) tut-

Comando/Sintassi	Cosa fa
	tutte le occorrenze della stringa di ricerca e le elenca
`sed [opzioni] file`	editor di flusso per manipolare file da uno script o da linea di comando

7.2.1. grep

Questa sezione fornisce un'introduzione all'uso delle *espressioni regolari* con grep(1).

L'utility grep(1) viene usata per ricercare espressioni regolari comuni che si presentano nei file Unix. Le espressioni regolari, come quelle viste in precedenza, sono meglio specificate all'interno di apostrofi (o caratteri di quoting singoli) quando usate con l'utility grep(1). L'utility egrep(1) fornisce una capacità di ricerca attraverso un set esteso di meta-caratteri. La sintassi dell'utility grep(1), alcune delle possibili opzioni e alcuni semplici esempi sono mostrati di seguito.

Sintassi

```
grep [opzioni] expreg [file]
```

Opzioni generali

`-i`	ignora la differenza tra caratteri maiuscoli e minuscoli
`-c`	riporta solamente la somma del numero di linee contenenti le corrispondenze, non le corrispondenze stesse
`-v`	inverte la ricerca, visualizzando solo le linee senza corrispondenza
`-n`	mostra un numero di linea insieme alla linea su cui è stata trovata una corrispondenza
`-s`	lavora in silenzio, riportando solo lo stato finale:
	0, per corrispondenze trovate
	1, per nessuna corrispondenza
	2, per errori
`-l`	elenca i nomi dei file, ma non le linee, nei quali sono state trovate corrispondenze

Esempi:

Si consideri il seguente file:

```
{Unix prompt 5} cat num.list
 1      15 fifteen
 2      14 fourteen
 3      13 thirteen
 4      12 twelve
 5      11 eleven
 6      10 ten
 7       9 nine
 8       8 eight
 9       7 seven
```

```
10       6 six
11       5 five
12       4 four
13       3 three
14       2 two
15       1 one
```

Ecco alcuni esempi di grep(1) usando tale file. Nel primo si ricerca il numero *15*:

```
{Unix prompt 6} grep '15' num.list
 1      15 fifteen
15       1 one
```

Ora si usa l'opzione `-c` per contare il numero di linee che corrispondono al precedente criterio di ricerca:

```
{Unix prompt 7} grep -c '15' num.list
2
```

Qui la ricerca è più generale: si selezionano tutte le linee che contengono il carattere *1* seguito da un *1* o un *2* o un *5*:

```
{Unix prompt 8} grep '1[125]' num.list
 1      15 fifteen
 4      12 twelve
 5      11 eleven
11       5 five
12       4 four
15       1 one
```

Ora si ricercano tutte le linee che *iniziano* con uno *spazio*:

```
{Unix prompt 9} grep '^ ' num.list
 1      15 fifteen
 2      14 fourteen
 3      13 thirteen
 4      12 twelve
 5      11 eleven
 6      10 ten
 7       9 nine
 8       8 eight
 9       7 seven
```

Ora tutte le linee che *non iniziano* con uno *spazio*:

```
{Unix prompt 10} grep '^[^ -]' num.list
10       6 six
11       5 five
12       4 four
13       3 three
14       2 two
15       1 one
```

L'ultimo esempio può anche essere realizzato usando l'opzione `-v` insieme all stringa di ricerca originale, esempio:

```
{Unix prompt 11} grep -v '^ ' num.list
10       6 six
11       5 five
12       4 four
13       3 three
14       2 two
15       1 one
```

Ora si ricercano tutte le linee che *iniziano* con carattere *compreso* tra *1* e *9*:

```
{Unix prompt 12} grep '^[1-9]' num.list
10       6 six
11       5 five
12       4 four
13       3 three
14       2 two
15       1 one
```

In questo esempio si ricercano alcune istanze di *t* seguite da *zero o alcune* occorrenze di *e*:

```
{Unix prompt 13} grep 'te*' num.list
 1      15 fifteen
 2      14 fourteen
 3      13 thirteen
 4      12 twelve
 6      10 ten
 8       8 eight
13       3 three
14       2 two
```

In questo esempio si ricercano alcune istanze di *t* seguite da *una o alcune* occorrenze di e:

```
{Unix prompt 14} grep 'tee*' num.list
1      15 fifteen
2      14 fourteen
3      13 thirteen
6      10 ten
```

Si può prendere il proprio input da un programma, anzichè da un file. Qui si riportano alcune linee di output del comando who(1) che iniziano con la lettera *l*.

```
{Unix prompt 15} who | grep '^l'
lcondron ttyp0 Dec 1 02:41 (lcondron-pc.acs.)
```

7.2.2. sed

L'editor di flusso non interattivo sed(1) manipola un flusso di input, linea per linea, creando specifici cambiamenti e mandando il risultato su standard output.

Sintassi

```
sed [opzioni] comando_di_editing [file]
```

Il formato per i comandi di editing è:

```
[indirizzo1[,indirizzo2]] [funzione] [argomenti]
```

dove gli indirizzi sono facoltativi e possono essere separati dalla funzione tramite spazi o tab. La funzione è obbligatoria. L'argomento può essere facoltativo o obbligatorio a seconda della funzione usata.

Gli *indirizzi di linea numerati* sono numeri decimali di linea che partono dalla prima linea di input e si incrementano di uno per ogni linea. Se vengono stabiliti più file di input il contatore continua cumulativamente attraverso i file. L'ultima linea di input può essere specificata con il carattere `$`.

Gli *indirizzi di contesto* sono schemi di espressioni regolari racchiusi tra caratteri di slashe (/).

I comandi possono avere 0, 1 o 2 indirizzi separati da virgola con i seguenti effetti:

# indirizzi	linee considerate
0	tutte le linee di input
1	solamente le linee che corrispondono agli indirizzi specificati
2	dalla prima linea che corrisponde al primo indirizzo fino alla linea che corrisponde al secondo indirizzo, inclusa. Il processo viene ripetuto per le linee interne.

Le *funzioni di sostituzione* permettono di ricercare contesti e sono specificate nella forma:

```
s/schema_espressione_regolare/stringa_di_rimpiazzo/flag
```

e possono essere quotate con caratteri di quoting singoli (') se sono specificate opzioni o funzioni aggiuntive. Questi schemi sono identici agli indirizzi di contesto, eccetto che, mentre questi sono normalmente chiusi tra slashe (/), nelle funzioni sono permessi alcuni normali caratteri per specificare i delimitatori, oltre a newline e spazio. La stringa di rimpiazzo non è uno schema di espressione regolare; qui i caratteri non hanno significati speciali, fatta eccezione di:

&	che sostituisce tale simbolo con la stringa *`schema_espressione_regolare`*
\n	sostituisce tale simbolo con l'*`n-esima`* stringa corrispondente a *`schema_espressione_regolare`* chiusa tra una coppia di '\(','\)'

Questi caratteri speciali possono essere messi in escape con il carattere backslash (\) per rimuovere il loro significato speciale.

Opzioni generali

`-e` *`script`*	script di editing
`-n`	non stampa l'output di default, ma solamente quelle linee specificate dalle funzioni `p` o `s///p`
`-f` *`script_file`*	prende lo script di editing dal file specificato

Alcune valide flag per le funzioni sostitutive sono:

`d`	cancella lo schema
`g`	sostituzione globale dello schema
`p`	stampa le linee

Esempi:

Questo esempio modifica tutte le accidentali virgole (,) in una virgola seguita da uno spazio (,) quindi crea l'output:

```
% cat filey | sed s/,/,\ /g
```

Il seguente esempio rimuove tutte le accidentali *Jr* precedute da uno spazio (*Jr*) all'interno del file `filey`:

```
% cat filey | sed s/\ Jr//g
```

Per realizzare operazioni multiple sull'input, si precede ogni operazione con l'opzione `-e` (edit) e si quota la stringa. Ad esempio, per filtrare le linee contenenti «Date: » e «From: » e rimpiazzarle senza i due punti (:):

```
% sed -e 's/Date: /Date /' -e 's/From: /From /'
```

Per visualizzare solamente le linee del file che iniziano con «Date:» e includerne una che inizia con «Name:»:

```
% sed -n '/^Date:/,/^Name:/p'
```

Per stampare solamente le prime 10 linee dell'input (un rimpiazzo di head(1)):

```
% sed -n 1,10p
```

7.2.3. awk, nawk, gawk

awk(1) è un linguaggio di elaborazione e ricerca di schemi. Il suo nome deriva dalle ultime iniziali dei tre autori: Alfred. V. Aho, Peter. J.Weinberger e Brian. W. Kernighan. `nawk` è

un *nuovo* awk(1), una nuova versione del programma e gawk(1) è il *gnu* awk(1), da parte della Free Software Foundation. Ogni versione è leggermente differente. Qui ci si limiterà ad illustrare semplici esempi che potrebbero andar bene per tutte le versioni. In alcuni sistemi operativi awk(1) è in realtà `nawk`.

awk(1) ricerca schemi nel suo input e realizza le operazioni specificate su ogni linea o sui campi di linea che contengono tali schemi. Le espressioni dello schema di confronto per awk(1) possono essere specificate sia attraverso linea di comando, sia inserendole in un file e usando l'opzione `-f` *`file_programma`* .

Sintassi

```
awk programma [file]
```

dove *`programma`* è composto da uno o più dei seguenti campi:

`schema` { *`azione`* }

Ogni linea di input viene verificata con lo schema di confronto insieme alla specifica azione che bisogna realizzare per ogni corrispondenza trovata. Questo continua attraverso la completa sequenza di schemi, quindi la prossima linea di input viene verificata.

L'*input* è diviso tra *record* e *campi*. Il separatore di *record* di default è newline e la variabile `NR` tiene il conto dei record. Il separatore di *campo* di default è uno spazio bianco, *spazi* e *tab*, e la variabile `NF` tiene il conto dei campi. I separatori di input del campo, `FS` e del record, `RS`, possono essere settati in qualsiasi momento per farli corrispondere a singoli caratteri specifici. I separatori di output del campo, `OFS` e del record, `ORS`, possono essere modificati, se si desidera, con singoli caratteri specifici. `$n`, dove *n* è un intero, viene usato per rappresentare l'*`n-esimo`* campo di un record di input, mentre `$0` rappresenta l'intero record di input.

`BEGIN` e `END` sono speciali schemi che vengono verificati rispettivamente all'inizio dell'input, prima che il primo campo sia letto e alla fine dell'input, dopo che l'ultimo campo è stato letto.

La *stampa* è permessa attraverso l'istruzione `print` e l'istruzione per la stampa formattata `printf`.

Gli *schemi* (pattern) possono essere espressioni regolari, espressioni aritmetiche relazionali, espressioni di valutazione di stringhe e combinazioni buleane di alcune di queste. In quest'ultimo caso gli schemi possono essere combinati con i seguenti operatori buleani, usando le parentesi per definire le combinazioni:

||	or
&&	and
!	not

La separazione di schemi con virgole definisce un *range* in cui lo schema è applicabile, esempio:

```
/primo/,/ultimo/
```

seleziona tutte le linee partendo con quella che contiene *primo* e continuando inclusivamente fino alla linea che contiene *ultimo*.

Per selezionare le linee da 15 a 20 si usa il seguente schema:

NR==15 , NR==20

Le *espressioni regolari* devono essere chiuse tra slashe (/) e i meta-caratteri possono essere messi in escape con il carattere di backslash (\). Le espressioni regolari possono essere raggruppate con gli operatori seguenti:

|	per alternative separate
+	una o più
?	zero o una

Un confronto di espressione regolare può essere specificato con:

~	contiene l'espressione
!~	non contiene l'espressione

Quindi il programma:

```
$1 ~ /[Ff]rank/
```

è vero se il primo campo, $1, contiene "Frank" o "frank" dovunque all'interno del campo. Per confrontare un campo identico a "Frank" o "frank" si usa:

```
$1 ~ /^[Ff]rank$/
```

Le *espressioni relazionali* sono permesse usando i seguenti operatori relazionali:

<	minore di
<=	minore o uguale a
= =	uguale a
>=	maggiore o uguale a
!=	non uguale a
>	maggiore di

Non si può conoscere su due piedi se le variabili sono stringhe o numeri. Se nessun operando è riconosciuto per essere un numero, sono realizzati confronti di stringhe. Altri-

menti, viene realizzata una comparazione numerica. In mancanza di informazioni per il contrario, viene realizzata una comparazione di stringa, così questa:

```
$1 > $2
```

verrà valutata con valori di tipo stringa. Per assicurarsi una valutazione numerica, costruire qualcosa simile a:

```
( $1 + 0 ) > $2
```

Le *funzioni matematiche* exp, log e sqrt sono di tipo built-in.

Altre funzioni *built-in* sono:

index(s,t)	ritorna la posizione della stringa *s* dove si presenta il primo *t* o 0 se non esiste
lenght(s)	ritorna la lunghezza della stringa *s*
substr(s,m,n)	ritorna l'*n-esimo* carattere della sottostringa di *s*, iniziando dalla posizione *m*

Gli *array* sono dichiarati automaticamente quando vengono usati, per esempio:

```
arr[i]=$1
```

assegna il primo campo del corrente record di input all'i-esimo elemento dell'array.

Le espressioni di controllo di flusso *if-else*, *while* e *for* sono permesse con la sintassi del *C*:

```
for (i=1; i <= NF; i++) {azioni}

while (i<=NF) {azioni}

if (i<NF) {azioni}
```

Opzioni generali

`-f` *file_programma*	legge i comandi dal file specificato
`-F`*c*	usa il carattere *c* come il carattere di separatore di campo

Esempi:

```
% cat filex | tr a-z A-Z | awk -F: '{printf("7R %-6s %-9s %-24s \n",$1,
$2,$3)}' > upload.file
```

effettua `cat` su `filex`, che è formattato in questo modo:

```
nfb791:99999999:smith
```

```
7ax791:999999999:jones
8ab792:99999999:chen
8aa791:999999999:mcnulty
```

cambiando tutti i caratteri minuscoli in caratteri maiuscoli con l'utility tr(1) e formattando il file come mostrato di seguito, il quale viene scritto nel file `upload.file`

```
7R NFB791 99999999 SMITH
7R 7AX791 999999999 JONES
7R 8AB792 99999999 CHEN
7R 8AA791 999999999 MCNULTY
```

Capitolo 8. Altri comandi utili

8.1. Lavorare con i file

Questa sezione descrive alcuni comandi che possono risultare utili nell'esaminare e manipolare il contenuto dei propri file.

Tabella 8.1. Utilità file

Comando/Sintassi	Cosa fa
`cmp [opzioni] file1 file2`	confronta due file e mostra dove avvengono le differenze (file di testo e file binari)
`cut [opzioni] [file]`	taglia specifici campi/caratteri dalle linee di un file
`diff [opzioni] file1 file2`	confronta due file e mostra le differenze (solamente file di testo)
`file [opzioni] file`	classifica il tipo di file
`find directory [opzioni] [azioni]`	cerca file basandosi sul tipo o su uno schema
`ln [opzioni] sorgente destinazione`	crea un link (collegamento) a `sorgente` chiamato `destinazione`
`paste [opzioni] file`	aggiunge campi all'interno di un file
`sort [opzioni] file`	riordina le linee di un file in accordo con le opzioni specificate
`strings [opzioni] file`	riporta sequenze di 4 o più caratteri stampabili terminati con <NL> o <NULL>. Normalmente utilizzato per ricercare stringhe ASCII in file binari.
`tee [opzioni] file`	copia standard output in uno o più file

Comando/Sintassi	Cosa fa
`touch [opzioni] [data/ora] file`	crea un file vuoto o aggiorna la data di accesso di un file esistente
`tr [opzioni] stringa1 stringa2`	traduce i caratteri di *`stringa1`* provenienti da standard input in quelli di *`stringa2`* per standard output
`uniq [opzioni] file`	rimuove le linee ripetute in un file
`wc [opzioni] [file]`	mostra il numero di parole (o di caratteri o di linee) di un file

8.1.1. cmp - confronta contenuti di file

Il comando cmp(1) confronta due file, e (senza opzioni) riporta la posizione della loro prima differenza. Può trattare confronti sia tra file ASCII sia tra file binari. Il comando compie una comparazione byte-per-byte.

Sintassi

```
cmp [opzioni] file1 file2 [salto1] [salto2]
```

I numeri di *salto* sono i numeri di byte da saltare in ogni file prima di iniziare il confronto.

Opzioni generali

`-l`	riporta ogni differenza
`-s`	riporta solamente un valore di uscita, non le differenze tra i byte

Esempi:

Dati i file `mon.logins` e `tues.logins`:

```
ageorge         ageorge
bsmith          cbetts
cbetts          jchen
jchen           jdoe
jmarsch         jmarsch
lkeres          lkeres
mschmidt        proy
sphillip        sphillip
wyepp           wyepp
```

Il confronto dei due file produce:

```
% cmp mon.logins tues.logins
```

```
mon.logins tues.logins differ: char 9, line 2
```

Di default il comando riporta solamente la prima differenza trovata.

Questo comando è utile nel determinare quale versione di un file dovrebbe essere mantenuta quando c'è più di una versione dello stesso file.

8.1.2. diff - differenze tra file

Il comando diff(1) confronta due file, due directory, ecc., e riporta tutte le differenze tra i due. Questo comando tratta solamente file ASCII. Il suo formato di output è stato progettato per fornire i cambiamenti necessari per convertire il primo file nel secondo.

Sintassi

```
diff [opzioni] file1 file2
```

Opzioni generali

`-b`	ignora spazi contigui
`-i`	ignora la differenza tra lettere minuscole e maiuscole
`-w`	ignora i caratteri di spazio e tab
`-e`	produce un formato di output da utilizzare con l'editor ed(1)
`-r`	esegue diff(1) ricorsivamente attraverso le sotto-directory

Esempi:

Per i file sopra citati `mon.logins` e `tues.logins` , le loro differenze sono:

```
% diff mon.logins tues.logins
2d1
< bsmith
4a4
> jdoe
7c7
< mschmidt
---
> proy
```

Notare che l'output mostra sia l'elenco delle differenze sia in quale file queste esistono. Le linee del primo file sono precedute da < e quelle del secondo file sono precedute da >.

8.1.3. cut - seleziona parte di una linea di un file

Il comando cut(1) permette di estrarre una parte di un file che verrà utilizzata per un altro scopo.

Sintassi

```
cut [opzioni] file
```

Opzioni generali

`-c` *`lista_caratteri`*	posizioni dei caratteri da selezionare (il primo carattere è in posizione 1)
`-d` *`delimitatore`*	delimitatore di campo (tab di default)
`-f` *`lista_campi`*	campi da selezionare (il primo campo è 1)

Sia la lista dei caratteri che quella dei campi possono contenere numeri (in ordine crescente) separati da virgole o da spazi bianchi e possono contenere un trattino (-) per indicare un range. La mancanza di un numero prima del trattino (esempio -5) o dopo il trattino (esempio 5-), specifica rispettivamente un range completo che inizia con il primo carattere o campo, o finisce con l'ultimo carattere o campo. I caratteri bianchi di separazione di lista devono essere chiusi tra caratteri di quoting. I campi di delimitazione possono essere chiusi tra quoting se hanno un significato speciale di shell, ad esempio quando specificano un carattere di tab o spazio.

Esempi:

In questi esempi si userà il file `users`:

```
jdoe    John Doe        4/15/96
lsmith  Laura Smith     3/12/96
pchen   Paul Chen       1/5/96
jhsu    Jake Hsu        4/17/96
sphilip Sue Phillip     4/2/96
```

Se si desidera solo lo username e il nome reale dell'utente, il comando cut(1) può essere usato per ottenere solamente queste informazioni:

```
% cut -f 1,2 users
jdoe    John Doe
lsmith  Laura Smith
pchen   Paul Chen
jhsu    Jake Hsu
sphilip Sue Phillip
```

Il comando cut(1) può essere usato con altre opzioni. L'opzione `-c` permette di selezionare caratteri. Per selezionare i primi 4 caratteri:

```
% cut -c 1-4 users
```

che produrrà:

```
jdoe
lsmi
pche
jhsu
sphi
```

in questo modo si selezionano solamente i primi 4 caratteri di ogni linea.

8.1.4. paste - fusione di file

Il comando paste(1) permette di combinare insieme due file. In una fusione, il delimitatore di default tra le colonne è un tab, ma le opzioni permettono di usare altri delimitatori.

Sintassi

```
paste [opzioni] file1 file2
```

Opzioni generali

`-d` *`lista`*	elenco dei caratteri di delimitazione
`-s`	concatena linee

L'elenco dei *delimitatori* può comprendere singoli caratteri come una virgola, una stringa quotata, uno spazio o alcune delle seguenti sequenze di escape:

\n	carattere newline
\t	carattere tab
\\	carattere backslash
\0	stringa vuota (carattere non-null)

Può essere necessario quotare i delimitatori che hanno significati speciali di shell.

Un trattino (-) al posto di un nome di file viene usato per indicare che quel campo dovrebbe venire da standard input.

Esempi:

Dato il file `users`:

```
jdoe    John Doe        4/15/96
lsmith  Laura Smith     3/12/96
pchen   Paul Chen       1/5/96
jhsu    Jake Hsu        4/17/96
sphilip Sue Phillip     4/2/96
```

e il file phone:

```
John Doe        555-6634
Laura Smith     555-3382
Paul Chen       555-0987
Jake Hsu        555-1235
Sue Phillip     555-7623
```

il comando paste(1) può essere usato in combinazione con il comando cut(1) per creare un nuovo file, `listing`, che include per tutti gli utenti lo username, il nome reale, l'ultimo

login e il numero di telefono. Prima si estraggono i numeri di telefono all'interno del file temporaneo `temp.file` :

```
% cut -f2 phone > temp.file
555-6634
555-3382
555-0987
555-1235
555-7623
```

Il risultato può essere incollato alla fine di ogni linea di `users` e quindi rediretto nel nuovo file, `listing`:

```
% paste users temp.file > listing
jdoe    John Doe        4/15/96         237-6634
lsmith  Laura Smith     3/12/96         878-3382
pchen   Paul Chen       1/5/96          888-0987
jhsu    Jake Hsu        4/17/96         545-1235
sphilip Sue Phillip     4/2/96          656-7623
```

Tutto questo può anche essere realizzato su una linea senza il file temporaneo, con lo stesso risultato:

```
% cut -f2 phone | paste users - > listing
```

In questo caso il trattino (-) funge come sostituto per il campo di input (indica cioè l'output del comando cut(1)).

8.1.5. touch - crea un file

Il comando touch(1) può essere usato per creare un nuovo (vuoto) file o per aggiornare l'ultima data/ora di accesso di un file esistente. Il comando viene usato primariamente quando uno script richiede la pre-esistenza di un file (ad esempio per appendere delle informazioni) o quando uno script controlla l'ultima data/ora in cui una funzione è stata realizzata.

Sintassi

```
touch [opzioni] [data/ora1] file

touch [opzioni] [-t data/ora2] file
```

Opzioni generali

`-a`	cambia la data/ora di accesso al file (solamente SVR4)
`-c`	non crea il file se non esiste
`-f`	forza l'azione, nonostante i permessi di lettura/scrittura del file
`-m`	cambia la data/ora di modifica del file (solamente SVR4)

`-t data/ora2`	usa la data/ora2 specificata, non la data/ora corrente (solamente SVR4)

Quando si usa l'opzione `-t data/ora2`, deve essere nella forma:

[[*CC*]*YY*] *MMDDhhmm* [.*SS*]

dove:

CC	prime due cifre dell'anno
YY	seconde due cifre dell'anno
MM	mese, 01-12
DD	giorno del mese, 01-31
hh	ora del giorno, 00-23
mm	minuti, 00-59
SS	secondi, 00-59

Il formato dell'opzione *data/ora1* è:

MMDDhhmm [*YY*]

dove questi simboli hanno gli stessi significati dei simboli mostrati in precedenza.

La data non può essere settata prima del 1969 o dopo il 18 Gennaio 2038 (dipende dalle versione del sistema operativo Unix).

Esempi:

Per creare un file:

```
% touch filename
```

8.1.6. wc - conta le parole in un file

wc(1) sta per «conta parole»; il comando può essere usato per contare il numero di linee, di caratteri o di parole in un file.

Sintassi

```
wc [opzioni] file
```

Opzioni generali

`-c`	conta byte
`-m`	conta caratteri (SVR4)
`-l`	conta linee

`-w`	conta parole

Se nessuna opzione viene specificata, l'opzione di default è `-lwc`.

Esempi:

Dato il file `users`:

```
jdoe    John Doe        4/15/96
lsmith  Laura Smith     3/12/96
pchen   Paul Chen       1/5/96
jhsu    Jake Hsu        4/17/96
sphilip Sue Phillip     4/2/96
```

il risultato dell'uso del comando wc(1) su tale file è il seguente:

```
% wc users
5 20 121 users
```

Il primo numero indica il numero di linee nel file, il secondo indica il numero di parole e il terzo numero indica il numero di caratteri.

Usando il comando wc(1) con una delle opzioni sopra citate (`-l` per linee; `-w` per parole o `-c` per caratteri) si ottiene come risultato solo una delle precedenti informazioni. Per esempio, `wc -l users` produce il risultato seguente:

```
5 users
```

8.1.7. ln - crea un link a un altro file

Il comando ln(1) crea un «link» (collegamento) o un modo aggiuntivo per accedere (o attribuisce un nome addizionale) a un altro file.

Sintassi

`ln [opzioni] sorgente [destinazione]`

Se non specificata, la `destinazione` di default è un file dello stesso nome di `sorgente` posto nella directory di lavoro corrente.

Opzioni generali

`-f`	forza un link nonostante i permessi della destinazione; non riporta errori (solamente SVR4)
`-s`	crea un link simbolico

Esempi:

Un *link simbolico* viene usato per creare un nuovo percorso a un altro file o directory. Per esempio, se un gruppo di utenti è abituato ad usare un comando chiamato `chkmag`, ma il

comando è stato riscritto e il nuovo nome è `chkit`, creando un link simbolico gli utenti eseguiranno automaticamente `chkit` quando digitano il comando `chkmag`, eliminando la transizione per il nuovo comando.

Un link simbolico può essere creato nel modo seguente:

```
% ln -s chkit chkmag
```

Ora il lungo listato per questi due file è il seguente:

```
16 -rwxr-x--- 1 lindadb acs 15927 Apr 23 04:10 chkit
1  lrwxrwxrwx 1 lindadb acs 5     Apr 23 04:11 chkmag -> chkit
```

Notare che mentre i permessi di `chkmag` sono aperti a tutti, poichè è linkato a `chkit`, le caratteristiche dei permessi, del gruppo e del proprietario di `chkit` saranno rispettate quando `chkmag` viene invocato.

Con un link simbolico, il link può esistere senza che il file o la directory a cui è collegato esista.

Un *hard link* può solamente essere creato per un altro file sullo stesso file system, ma non per una directory (eccetto per il super-user). Un hard link crea un nuovo elemento di directory puntante allo stesso inode del file originale. Il file linkato deve esistere prima che l'hard link possa essere creato. Il file non sarà cancellato fino a quando tutti gli hard link saranno rimossi. Per linkare i due file precedenti tramite un hard link:

```
% ln chkit chkmag
```

Quindi un lungo listato mostra che il numero di *inode* (742) è lo stesso per entrambi i file:

```
% ls -il chkit chkmag
742 -rwxr-x--- 2 lindadb acs 15927 Apr 23 04:10 chkit
742 -rwxr-x--- 2 lindadb acs 15927 Apr 23 04:10 chkmag
```

8.1.8. sort - ordina il contenuto di un file

Il comando sort(1) viene usato per ordinare le linee di un file. Si possono usare diverse opzioni per stabilire il modo di ordinamento e su quali campi ordinare il file. Senza opzioni, sort(1) confronta intere linee di un file e produce un ordinamento ASCII (prima i numeri, lettere maiuscole e quindi lettere minuscole).

Sintassi

```
sort [opzioni] [+pos1 [-pos2]] file
```

Opzioni generali

`-b`	ignora gli spazi bianchi iniziali (spazi e tab) quando si stabiliscono i carattere di inizio e di fine per la chiave di ordinamento

`-d`	ordinamento a dizionario, sono rilevanti solamente le lettere, le cifre, spazi e tab
`-f`	uguaglia lettere maiuscole e minuscole
`-k` *`chiave`*	ordinamento su chiavi specifiche (non disponibile su tutti i sistemi)
`-i`	ignora i caratteri non stampabili
`-n`	ordinamento numerico
`-o` *`outfile`*	file di output
`-r`	ribalta l'ordine
`-t` *`car`*	usa *`car`* come il carattere di separatore di campo
`-u`	unico; trascura multiple copie di linee uguali (dopo l'ordinamento)
`+pos1` `[-`*`pos2`*`]`	(vecchio stile) fornisce funzionalità simili all'opzione `-k` *`chiave`*.

Per gli elementi di *posizione* (`+/-`), *`pos1`* è il numero di parola di inizio, iniziando da *0* e *`pos2`* è il numero di parola di fine. Quando *`-pos2`* non è specificato, il campo di ordinamento continua fino alla fine del file. Sia *`pos1`* che *`pos2`* possono essere specificati nella forma *w.c*, dove *w* è il numero di parola e *c* è il carattere all'interno della parola. Per *c 0* si specifica il delimitatore che precede il primo carattere e *1* è il primo carattere della parola. Questi elementi possono essere seguiti da un tipo di modificatore, esempio *n* per numerico, *b* per saltare gli spazi bianchi ,ecc.

Il campo *`chiave`* dell'opzione `-k` ha la seguente sintassi:

`campo_inizio` `[`*`tipo`*`][,`*`campo_fine`* `[`*`tipo`*`]]`

dove:

`campo_inizio, campo_fine`	definiscono le chiavi per restringere l'ordinamento su una porzione di linea
`tipo`	modifica l'ordinamento: validi modificatori sono dati dai singoli caratteri (bdfiMnr) derivanti dalle simili opzioni di ordinamento, ad esempio un tipo *b* equivale a `-b`, ma applicato solamente nello specifico campo di azione

Esempi:

Dato il file `users`:

```
jdoe    John Doe        4/15/96
lsmith  Laura Smith     3/12/96
pchen   Paul Chen       1/5/96
jhsu    Jake Hsu        4/17/96
sphilip Sue Phillip     4/2/96
```

ordinando con sort(1) gli utenti si produce:

```
jdoe    John Doe        4/15/96
jhsu    Jake Hsu        4/17/96
lsmith  Laura Smith     3/12/96
pchen   Paul Chen       1/5/96
sphilip Sue Phillip     4/2/96
```

Se, tuttavia, si desidera un elenco in ordine di nome, si usa l'opzione per specificare su quale campo ordinare (i campi sono numerati partendo da 0):

```
% sort +2 users
pchen   Paul Chen       1/5/96
jdoe    John Doe        4/15/96
jhsu    Jake Hsu        4/17/96
sphilip Sue Phillip     4/2/96
lsmith  Laura Smith     3/12/96
```

Per ribaltare l'ordine:

```
% sort -r users
sphilip Sue Phillip     4/2/96
pchen   Paul Chen       1/5/96
lsmith  Laura Smith     3/12/96
jhsu    Jake Hsu        4/17/96
jdoe    John Doe        4/15/96
```

Un'opzione particolarmente utile di ordinamento è l'opzione `-u`, che elimina gli elementi duplicati nel file mentre si ordina il file. Per esempio, il file `todays.logins` :

```
sphillip
jchen
jdoe
lkeres
jmarsch
ageorge
lkeres
proy
jchen
```

mostra una lista di ogni username che ha effettuato il login nel sistema in giornata. Se si vuole conoscere quanti unici utenti abbiano effettuato il login nel sistema in giornata, usando sort(1) con l'opzione `-u`, la lista conterrà ciascun utente una volta sola. (Il comando può essere mandato in pipe a `wc -l` per ottenere direttamente il numero):

```
% sort -u todays.logins
```

```
ageorge
jchen
jdoe
jmarsch
lkeres
proy
sphillip
```

8.1.9. tee - copia l'output di un comando

Il comando tee(1) manda lo standard input nel file specificato e anche nello standard output. Viene spesso usato in una pipe di comandi.

Sintassi

```
tee [opzioni] [file]
```

Opzioni generali

`-a`	appende l'output ai file
`-i`	ignora gli interrupt

Esempi:

In questo primo esempio l'output di who(1) è visualizzato sullo schermo e memorizzato nel file `users.file` :

```
brigadier: condron [55]> who | tee users.file
condron ttyp0 Apr 22 14:10 (lcondron-pc.acs.)
frank   ttyp1 Apr 22 16:19 (nyssa)
condron ttyp9 Apr 22 15:52 (lcondron-mac.acs)
```

```
brigadier: condron [56]> cat users.file
condron ttyp0 Apr 22 14:10 (lcondron-pc.acs.)
frank   ttyp1 Apr 22 16:19 (nyssa)
condron ttyp9 Apr 22 15:52 (lcondron-mac.acs)
```

Nel prossimo esempio l'output di who(1) viene mandato nei file `users.a` e `users.b` . Viene anche mandato in pipe con il comando wc(1), che riporta il numero di linee.

```
brigadier: condron [57]> who | tee users.a users.b | wc -l
3
```

```
brigadier: condron [58]> cat users.a
condron ttyp0 Apr 22 14:10 (lcondron-pc.acs.)
frank   ttyp1 Apr 22 16:19 (nyssa)
condron ttyp9 Apr 22 15:52 (lcondron-mac.acs)
```

Nell'esempio seguente un listato lungo di directory viene mandato nel file `files.long` . Inoltre viene messo in pipe con il comando grep(1) che riporta quali file sono stati modificati in Agosto.

```
brigadier: condron [60]> ls -l | tee files.long |grep Aug
1 drwxr-sr-x 2 condron 512  Aug 8 1995 News/
2 -rw-r--r-- 1 condron 1076 Aug 8 1995 magnus.cshrc
2 -rw-r--r-- 1 condron 1252 Aug 8 1995 magnus.login
```

```
brigadier: condron [63]> cat files.long
total 34
2 -rw-r--r-- 1 condron 1253 Oct 10 1995 #.login#
1 drwx------ 2 condron 512  Oct 17 1995  Mail/
1 drwxr-sr-x 2 condron 512  Aug 8  1995  News/
5 -rw-r--r-- 1 condron 4299 Apr 21 00:18 editors.txt
2 -rw-r--r-- 1 condron 1076 Aug 8  1995  magnus.cshrc
2 -rw-r--r-- 1 condron 1252 Aug 8  1995  magnus.login
7 -rw-r--r-- 1 condron 6436 Apr 21 23:50 resources.txt
4 -rw-r--r-- 1 condron 3094 Apr 18 18:24 telnet.ftp
1 drwxr-sr-x 2 condron 512  Apr 21 23:56 uc/
1 -rw-r--r-- 1 condron 1002 Apr 22 00:14 uniq.tee.txt
1 -rw-r--r-- 1 condron 1001 Apr 20 15:05 uniq.tee.txt~
7 -rw-r--r-- 1 condron 6194 Apr 15 20:18 Unixgrep.txt
```

8.1.10. uniq - rimuove linee duplicate

Il comando uniq(1) filtra le linee adiacenti duplicate in un file.

Sintassi

`uniq [opzioni] [+|-n] file [nuovo.file]`

Opzioni generali

`-d`	solo una copia delle linee ripetute
`-u`	seleziona solamente le linee non ripetute
+n	ignora i primi *n* caratteri
`-s` *n*	come sopra (solamente SVR4)
-n	salta i primi *n* campi, spazi inclusi (spazi e tab)
`-f` *campi*	come sopra (solamente SVR4)

Esempi:

Si consideri il seguente file e l'esempio, dove uniq(1) rimuove la quarta linea da `file` e mette il risultato nel file `nuovo.file` .

```
{Unix prompt 1} cat file
1 2 3 6
4 5 3 6
7 8 9 0
7 8 9 0
```

```
{Unix prompt 2} uniq file nuovo.file
```

```
{Unix prompt 3} cat nuovo.file
1 2 3 6
4 5 3 6
7 8 9 0
```

Qui sotto, l'opzione *-n* del comando uniq(1) viene usata per saltare i primi 2 campi nel file e per filtrare le linee di uscita che sono duplicate dal terzo campo in avanti.

```
{Unix prompt 4} uniq -2 file
1 2 3 6
7 8 9 0
```

8.1.11. strings - cerca stringhe ASCII

Per cercare stringhe ASCII, stampabili, in un file binario si usa il comando strings(1). Questo comando ricerca sequenze di quattro o più caratteri ASCII terminanti con un carattere newline o con un carattere null. strings(1) risulta utile per ricercare nomi di file e possibili messaggi di errore all'interno di un programma compilato, di cui non si ha il codice sorgente.

Sintassi

```
strings [opzioni] file
```

Opzioni generali

`-n` *numero*	usa *numero* come la minima lunghezza della stringa, piuttosto di 4 (solamente SVR4)
-numero	come sopra
`-t` *formato*	fa precedere alla stringa l'offset di byte dall'inizio del file; *formato* può essere: *d* = decimale, *o* = ottale *x* = esadecimale (solamente SVR4)
`-o`	fa precedere alla stringa l'offset di byte in decimale (solamente BSD)

Esempi:

```
% strings /bin/cut
SUNW_OST_OSCMD
nessun delimitatore specificato
delimitatore non valido
b:c:d:f:ns
cut: -n può solo essere usato con -b
cut: -d può solo essere usato con -f
cut: -s può solo essere usato con -f
nessuna lista specificata
cut: %s non si riesce ad aprire
il range specificato non è valido
troppi range specificati
i range devono essere crescenti
carattere non valido nel range
```

```
Errore interno nel processare l'input
carattere multibyte non valido
incapace di allocare memoria sufficiente
incapace di allocare memoria sufficiente
cut:
uso: cut -b lista [-n] [filename ...-]
     cut -c lista [filename ...-]
     cut -f lista [-d delim] [-s] [filename]
```

8.1.12. file - mostra il tipo di file

Il comando file(1) esamina il file specificato e cerca di determinare che tipo di file esso sia. Questo viene fatto leggendo i primi byte del file e confrontandoli con la tabella `/etc/magic`. Il comando file(1) può determinare file di testo ASCII, file formattati tar, file compressi, ecc.

Sintassi

```
file [opzioni] [-m file_magico] [-f lista_file] file
```

Opzioni generali

`-c`	verifica il file magico per eventuali errori nel formato
`-f` *`lista_file`*	`lista_file` contiene la lista dei file da esaminare
`-h`	non segue i link simbolici (solamente SVR4)
`-L`	segue i link simbolici (solamente BSD)
`-m` *`file_magico`*	usa `file_magico` come il file magico al posto di `/etc/magic`

Esempi:

Vengono elencati qui sotto gli output del comando `file filename` per alcuni file significativi.

```
/etc/magic:             File di testo ascii
/usr/local/bin/gzip:    Eseguibile paginato SPARC Sun linkato ↺
dinamicamente
/usr/bin/cut:           Eseguibile MSB a 32 bit ELF per SPARC ↺
versione 1, linkato dinamicamente, stripped
source.tar:             Archivio tar USTAR
source.tar.Z:           Blocchi di dati compressi 16 bit
```

8.1.13. tr - traduce caratteri

Il comando tr(1) traduce caratteri da stdin(4) a stdout(4).

Sintassi

```
tr [opzioni] stringa1 [stringa2]
```

Con nessuna opzione i caratteri di *stringa1* sono tradotti nei caratteri di *stringa2*, carattere per carattere, nell'ordine degli array di stringa. Il primo carattere di *stringa1* viene tradotto nel primo carattere di *stringa2*, ecc.

Un range di caratteri in una stringa viene specificato con un trattino tra il carattere superiore e quello inferiore del range, ad esempio per specificare tutte le lettere alfabetiche minuscole si usa *[a-z]*.

Ripetizioni di caratteri in *stringa2* possono essere rappresentate con la notazione *[x*n]*, dove il carattere *x* viene ripetuto *n* volte. Se *n* è *0* o assente, il carattere *x* viene assunto come valida corrispondenza richiesta in *stringa1*.

I caratteri possono includere le notazioni *ottale* (BSD e SVR4) e *carattere* (solamente SVR4). Qui *ottale* viene sostituito da una, due o tre cifre di interi ottali che codificano un carattere ASCII e *carattere* può essere uno di questi:

b	back space
f	form feed
n	newline
r	return
t	tab
v	tab verticale

La versione SVR4 di tr(1) permette l'operando *:classe*: nel campo stringa dove *classe* può assumere uno dei seguenti valori di classificazione carattere:

alpha	caratteri alfabetici
lower	caratteri alfabetici minuscoli
upper	caratteri alfabetici maiuscoli

Opzioni generali

`-c`	completa il set di caratteri in *stringa1*
`-d`	cancella i caratteri in *stringa1*
`-s`	modifica una stringa di caratteri ripetuti in *stringa1* in caratteri singoli

Esempi:

L'esempio seguente usa come file di input una lista di libri di P. G. Wodehouse Jeeves & Wooster.

```
The Inimitable Jeeves [1923]    The Mating Season [1949]
Carry On, Jeeves [1925]         Ring for Jeeves [1953]
```

```
Very Good, Jeeves [1930]        Jeeves and the Feudal Spirit [1954]
Thank You, Jeeves [1934]        Jeeves in the Offing [1960]
Right Ho, Jeeves [1934]         Stiff Upper Lip, Jeeves [1963]
The Code of the Woosters [1938] Much Obliged, Jeeves [1971]
Joy in the Morning [1946]       Aunts Aren't Gentlemen [1974]
```

Per tradurre tutte le lettere alfabetiche minuscole in maiuscole si possono usare:

```
tr '[a-z]' '[A-Z]'
```

oppure

```
tr '[:lower:]' '[:upper:]'
```

Affinchè tr(1) legga da stdin(4) si usa cat(1) in pipe con tr(1), così:

```
% cat wodehouse | tr '[a-z]' '[A-Z]'
THE INIMITABLE JEEVES [1923]    THE MATING SEASON [1949]
CARRY ON, JEEVES [1925]         RING FOR JEEVES [1953]
VERY GOOD, JEEVES [1930]        JEEVES AND THE FEUDAL SPIRIT [1954]
THANK YOU, JEEVES [1934]        JEEVES IN THE OFFING [1960]
RIGHT HO, JEEVES [1934]         STIFF UPPER LIP, JEEVES [1963]
THE CODE OF THE WOOSTERS [1938] MUCH OBLIGED, JEEVES [1971]
JOY IN THE MORNING [1946]       AUNTS AREN'T GENTLEMEN [1974]
```

Ora si cancellano tutti i numeri con:

```
% cat wodehouse | tr -d '[0-9]'
The Inimitable Jeeves []        The Mating Season []
Carry On, Jeeves []             Ring for Jeeves []
Very Good, Jeeves []            Jeeves and the Feudal Spirit []
Thank You, Jeeves []            Jeeves in the Offing []
Right Ho, Jeeves []             Stiff Upper Lip, Jeeves []
The Code of the Woosters []     Much Obliged, Jeeves []
Joy in the Morning []           Aunts Aren't Gentlemen []
```

Per trattenere tutte le occorrenze multiple dei caratteri e, r e f:

```
% cat wodehouse | tr -s 'erf'
The Inimitable Jeves [1923]     The Mating Season [1949]
Cary On, Jeves [1925]           Ring for Jeves [1953]
Very Good, Jeves [1930]         Jeves and the Feudal Spirit [1954]
Thank You, Jeves [1934]         Jeves in the Ofing [1960]
Right Ho, Jeves [1934]          Stif Upper Lip, Jeves [1963]
The Code of the Woosters [1938] Much Obliged, Jeves [1971]
Joy in the Morning [1946]       Aunts Aren't Gentlemen [1974]
```

8.1.14. find - cerca file

Il comando find(1) può cercare file in modo ricorsivo nell'albero di directory indicato, confrontando il tipo di file o lo schema specificato. find(1) può quindi elencare i file o eseguire arbitrari comandi in base ai risultati.

Sintassi

```
find directory [opzioni di ricerca] [azioni]
```

Opzioni generali

Per l'opzione del tempo di ricerca, la notazione in giorni, *n* è:

+n	più di *n* giorni
n	esattamente *n* giorni
-n	meno di *n* giorni

Alcune caratteristiche dei file che find(1) può ricercare sono:

la *data* dell'ultimo accesso o dell'ultima modifica al file:

`-atime` *n*	data di accesso, vero se l'accesso è di *n* giorni fà
`-ctime` *n*	data di cambiamento, vero se lo stato del file è stato modificato *n* giorni fà
`-mtime` *n*	data di modifica, vero se i dati del file sono stati modificati *n* giorni fà
`-newer` *filename*	vero se più recente del file specificato
`-type` *tipo*	*tipo* di *file*, dove *tipo* può essere:
b	file speciale a blocchi
c	file speciale a caratteri
d	directory
l	link simbolico
p	file pipe (fifo)
f	file regolare
`-fstype` *tipo*	*tipo* di *file system*, dove *tipo* può essere un valido tipo di file system, esempio *ufs* (Unix File System) e *nfs* (Network File System)
`-user` *username*	vero se il file appartiene all'utente specificato
`-group` *groupname*	vero se il file appartiene al gruppo specificato
`-perm [-]`*modi*	permessi del file, dove *modi* è la modalità ottale usata dal comando chmod(1). Quando *modi* è preceduto da un segno meno, solamente i bit che sono specificati vengono confrontati.

`-exec` *`comando`*	esegue il comando specificato. La fine del comando viene indicata con un escape e punto e virgola (\;). L'argomento del comando, {}, sostituisce il path name corrente.
`-name` *`filename`*	vero se il nome del file è quello specificato. Confronti con schemi wild card sono permessi se il meta-carattere viene messo in escape per la shell con un backslash (\)
`-ls`	sempre vero. Stampa una lista lunga del path name corrente
`-print`	stampa i path name trovati (default per SVR4, non per BSD)

Sono permesse espressioni complesse. Le espressioni devono essere raggruppate tra parentesi (mettendo in escape le parentesi con un backslash per impedire alla shell di interpretarle). Il simbolo di esclamazione (!) può essere usato per *negare* un'espressione. Gli operatori: `-a` (*and*) e `-o` (*or*) vengono usati per raggruppare espressioni.

Esempi:

find(1) può ricercare in modo ricorsivo attraverso sotto-directory, ma per lo scopo di questi esempi saranno usati i seguenti file:

```
14 -rw-r--r-- 1 frank staff  6682 Feb  5 10:04 library
 6 -r--r----- 1 frank staff  3034 Mar 16  1995 netfile
34 -rw-r--r-- 1 frank staff 17351 Feb  5 10:04 standard
 2 -rwxr-xr-x 1 frank staff   386 Apr 26 09:51 tr25*
```

Per trovare tutti i file più recenti del file `library`:

```
% find . -newer library -print
./tr25
./standard
```

Per trovare tutti i file con i permessi di lettura o esecuzione per l'utenza altri e quindi modificare gli stessi in modo da disabilitarli:

```
% find . \( -perm -004 -o -perm -001 \) -exec chmod o-rx {} \; -exec ls
 -al {} \;
-rw-r----- 1 frank staff  6682 Feb  5 10:04 ./library
-rwxr-x--- 1 frank staff   386 Apr 26 09:51 ./tr25
-rw-r----- 1 frank staff 17351 Feb  5 10:04 ./standard
```

Nell'esempio precedente le parentesi e i punti e virgola sono messi in escape con un backslash per impedire alla shell di interpretarli. Le parentesi graffe sono automaticamente rimpiazzate dai risultati della precedente ricerca e il punto e virgola chiude il comando.

Si possono ricercare alcuni nomi di file contenenti la stringa «ar» con:

```
% find . -name \*ar\* -ls
326584  7 -rw-r----- 1 frank staff  6682 Feb 5 10:04 ./library
```

```
326585 17 -rw-r----- 1 frank staff 17351 Feb 5 10:04 ./standard
```

dove l'opzione `-ls` mostra un listato lungo, numeri di inode inclusi.

8.2. Archiviazione, compressione e conversione di file

Tabella 8.2. Comandi di archiviazione, compressione e conversione di file

Comando/Sintassi	Cosa fa
`compress/uncompress/zcat [opzioni] file[.Z]`	comprime o decomprime un file. I file compressi sono memorizzati con l'estensione `.Z`
`dd [if=infile] [of=outfile] [operando=valore]`	copia un file, converte tra ASCII e EBCDIC o scambia l'ordine dei byte, come specificato
`gzip/gunzip/zcat [opzioni] file[.gz]`	comprime o decomprime un file. I file compressi sono memorizzati con l'estensione `.gz`
`od [opzioni] file`	effettua un dump di un file binario in ottale, in ASCII, in esadecimale, in decimale o in modo carattere.

Comando/Sintassi	Cosa fa
`tar [opzioni] [file]`	archivio a nastro - riferirsi alle pagine man per i dettagli su come creare, visualizzare ed estrarre un archivio di file. I file tar possono essere memorizzati su nastro o su disco.
`uudecode [file]`	decodifica un file uuencoded, ricreando il file originale
`uuencode [file] nuovo_nome`	codifica un file binario in ASCII 7-bit; utile quando lo si invia tramite email, per poi essere decodificato come `nuovo_nome` alla destinazione

8.2.1. Compressione di file

Il comando compress(1) viene usato per ridurre lo spazio di disco utilizzato da un file. Quando un file è stato compresso usando il comando compress(1), il suffisso `.Z` viene appeso al nome del file. I permessi, la data di accesso e di modifica del file originale vengono preservati. Il comando uncompress(1) restituisce il file originale compresso con compress(1).

Sintassi

```
compress [opzioni] [file]

uncompress [opzioni] [file.Z]

zcat [file.Z]
```

Opzioni generali

`-c`	scrive su standard output e non crea o modifica alcun file
`-f`	forza la compressione del file, anche se questa non riduce la dimensione del file o se il file destinazione (`file.Z`) esiste già.
`-v`	verbose. Riporta la percentuale di riduzione del file.

zcat(1) scrive su standard output. Equivale a `uncompress -c`.

Esempi:

Dati i file:

```
96 -rw-r--r-- 1 lindadb acs 45452 Apr 24 09:13 logins.beauty
184 -rw-r--r-- 1 lindadb acs 90957 Apr 24 09:13 logins.bottom
152 -rw-r--r-- 1 lindadb acs 75218 Apr 24 09:13 logins.photon
168 -rw-r--r-- 1 lindadb acs 85970 Apr 24 09:13 logins.top
```

Questi possono essere compressi con:

```
% compress logins.*
```

che crea i file:

```
24 -rw-r--r-- 1 lindadb acs  8486 Apr 24 09:13 logins.beauty.Z
40 -rw-r--r-- 1 lindadb acs 16407 Apr 24 09:13 logins.bottom.Z
24 -rw-r--r-- 1 lindadb acs 10909 Apr 24 09:13 logins.photon.Z
32 -rw-r--r-- 1 lindadb acs 16049 Apr 24 09:13 logins.top.Z
```

I file originali sono persi.

Per visualizzare un file compresso, viene usato il comando zcat(1):

```
% zcat logins.beauty.Z | head
beauty:01/22/94:#total logins,4338:#different UIDs,2290
beauty:01/23/94:#total logins,1864:#different UIDs,1074
beauty:01/24/94:#total logins,2317:#different UIDs,1242
beauty:01/25/94:#total logins,3673:#different UIDs,2215
beauty:01/26/94:#total logins,3532:#different UIDs,2216
beauty:01/27/94:#total logins,3096:#different UIDs,1984
beauty:01/28/94:#total logins,3724:#different UIDs,2212
beauty:01/29/94:#total logins,3460:#different UIDs,2161
```

```
beauty:01/30/94:#total logins,1408:#different UIDs,922
beauty:01/31/94:#total logins,2175:#different UIDs,1194
```

Una visualizzazione del file utilizzando un altro comando al posto di zcat(1) può produrre un difficile risultato binario.

Il comando uncompress(1) viene usato per far ritornare il file compresso nel suo formato originale:

```
% uncompress logins.*.Z -; ls -als logins.*
 96 -rw-r--r-- 1 lindadb acs 45452 Apr 24 09:13 logins.beauty
184 -rw-r--r-- 1 lindadb acs 90957 Apr 24 09:13 logins.bottom
152 -rw-r--r-- 1 lindadb acs 75218 Apr 24 09:13 logins.photon
168 -rw-r--r-- 1 lindadb acs 85970 Apr 24 09:13 logins.top
```

In aggiunta alle utility standard di Unix compress(1), uncompress(1), zcat(1) ci sono un set di utility *GNU* liberamente disponibili. Queste creano persino un miglior lavoro di compressione utilizzando un algoritmo più efficiente. I programmi GNU che forniscono funzionalità simili ai precedenti sono spesso installati rispettivamente come gzip(1), gunzip(1) e zcat(1). I nomi dei file compressi con gzip(1) hanno l'estensione `.z` o `.gz`. Il software GNU può essere ottenuto via ftp anonimo: `ftp://ftp.gnu.org/pub/gnu` .

8.2.2. tar - archivio di file

Il comando tar(1) raggruppa file in un dispositivo o in un file per scopi di archiviazione. Il comando tar(1) non comprime i file, rende solamente più maneggevole una grande quantità di file.

Sintassi

```
tar [opzioni] [directory file]
```

Opzioni generali

`c`	crea un archivio (inizia scrivendo all'inizio del file)
`t`	visualizza il contenuto
`x`	estrae da un archivio
`v`	verbose
`f`	nome del file di archivio
`b`	dimensione del blocco di archivio

tar(1), in generale, accetta le sue opzioni sia facendole precedere da un trattino (-) sia senza trattino. Il file di archivio può essere un file su disco, un dispositivo a nastro o standard input/output. L'ultimo modo viene rappresentato tramite un trattino.

Esempi:

Dati i file di dimensioni seguenti:

```
45 logs.beauty
89 logs.bottom
74 logs.photon
84 logs.top
```

tar(1) può combinare questi in un solo file, `logfile.tar` :

```
% tar -cf logfile.tar logs.* -; ls -s logfile.tar
304 logfile.tar
```

In Internet, molti siti di archivio FTP anonimi memorizzano i loro pacchetti in formati tar compressi, quindi i file finiscono in `.tar.Z` o `.tar.gz`. Per estrarre i file da uno di questi file archivio si può prima decomprimerlo, o usare l'appropriato comando zcat(1) e mandare in pipe il risultato a tar(1), esempio:

```
% zcat archive.tar.Z | tar -xvf -
```

dove il trattino alla fine del comando tar(1) indica che il file è preso da stdin(4).

8.2.3. uuencode/uudecode - codifica/decodifica un file

Per codificare un file binario in ASCII 7-bit si usa il comando uuencode(1). Per decodificare il file ASCII 7-bit in binario si usa il comando uudecode(1). Il suffisso «*uu*» nei nomi deriva dal fatto che questi comandi fanno parte del set di comandi Unix-to-Unix CoPy (UUCP). I comandi uuencode(1) e uudecode(1) sono generalmente usati quando si mandano file binari via e-mail. Con l'e-mail non c'è garanzia che file binari 8-bit siano trasferiti correttamente. Quindi per garantire una corretta consegna si può codificare il file binario, sia direttamente su linea di comando, includendo il file codificato, sia indirettamente, lasciando al proprio programma di posta MIME questo lavoro. In modo simile, l'utente decodifica il file ricevuto.

Sintassi

```
uuencode [file_sorgente] pathname_per_uudecode [> nuovo_file]
```

```
uudecode [-p] file_uuencode
```

Opzioni generali

`-p`	manda l'output su standard output piuttosto che nel file di default

Esempi:

La prima linea di un file codificato con uuencode(1) include i *permessi* e il *nome* del file originale che uudecode(1) utilizzerà quando decodificherà il file codificato. Il file inizia e finisce rispettivamente con le parole chiavi *begin* e *end*, esempio:

```
begin 555 binary_filename
```

```
M?T5,1@$" 0 " ( ! %"W #0 5"< T "
M!0 H !4 % 8 T $ - "@ H 4 P
M -0 !$ ! ! ! %"
M%P !0A< % $ $ 4(8 -"& W& W% < 0
M @ !0B T(@ )@ !P O=7-R+VQI8B]L9"YS
M;RXQ ?< 'Y VP "O !VP )8 &6 !G0
M %[ U0 %G !3 -;< #Q %Q !
MEP :P !_ '@ !PP (P
M N0 =H _0 $D Y < #F /L
M01 $' $ & ! P #0A@ 4(8
M" ! 0 !E !@ , T(@ %"( )@ $
M 0 ( -;@ $ ' -"N !0K@ /H
M $ # ', ! P #1J 4:@ #8 !
M !Y 0 , TH %* !=X 0
M@ @ # -/X !3^ "E, $ (4 !
M 4_> )0 0 ". P
; %0 P )@ $

end
```

8.2.4. dd - copia di blocchi e conversione

Il comando dd(1) permette di copiare dati da dispositivi grezzi, come dischi o nastri, specificando la dimensione dei blocchi di input e di output. dd(1) era originariamente conosciuto come il programma di copia disco-a-disco. Inoltre con dd(1) si possono convertire file in differenti formati, per esempio da EBCDIC a ASCII, cambiare l'ordine dei byte, ecc.

Sintassi

```
dd [if=dispositivo_input] [of=dispositivo_output] [operando=valore]
```

Opzioni generali

`if=`*`dispositivo_input`*	il dispositivo o file di input
`of=`*`dispositivo_output`*	il dispositivo o file di output

Se i dispositivi di input o di output non sono specificati, si assume come default rispettivamente lo standard input e lo standard output.

Gli operandi possono essere:

`ibs=`*`n`*	dimensione del blocco di input (blocchi da 512 byte di default)
`obs=`*`n`*	dimensione del blocco di output (blocchi da 512 byte di default)
`bs=`*`n`*	setta entrambe le dimensioni dei blocchi di input e di output

`files=`*n*	copia *n* file di input
`skip=`*n*	salta *n* blocchi di input prima di iniziare la copia
`count=`*n*	copia solamente *n* blocchi di input
`conv=`*valore* `[,`*valore*`]`	dove *valore* può essere:
`ascii`	converte da EBCDIC a ASCII
`ebcdic`	converte da ASCII a EBCDIC
`lcase`	converte i caratteri maiuscoli in minuscoli
`ucase`	converte i caratteri minuscoli in maiuscoli
`swab`	scambia ogni coppia di byte dei dati di input
`noerror`	non ferma il processo su un errore in input
`sync`	riempie ogni blocco di input alla dimensione di `ibs`, apporrendo byte nulli se necessario

La dimensione dei blocchi viene specificata in byte e può finire in *k*, *b* o *w* per indicare rispettivamente 1024 (kilo), 512 (blocco) o 2 (parola) byte.

Esempi:

Per copiare un file da un dispositivo a nastro in un altro:

```
% dd if=/dev/rmt/0 of=/dev/rmt/1
20+0 records in
20+0 records out
```

Per copiare file posti in un dispositivo a nastro, scritti su una macchina big endian con una dimensione di 20 blocchi, in file su una machina little endian che ha un nastro inserito nel suo dispositivo, si ha la necessità di scambiare le coppie di byte, in questo modo:

```
% dd if=/dev/rmt/0 of=new_file ibs=20b conv=swab
1072+0 records in
21440+0 records out
```

A completamento dell'operazione, dd(1) riporta il numero dei blocchi totali e dei blocchi parziali per entrambi i file di input e di output.

8.2.5. od - dump ottale di un file

Il comando od(1) effettua un dump di un file su stdout(4) in differenti formati, incluso l'ottale, il decimale, virgola mobile, esadecimale e formato carattere.

Sintassi

```
od [opzioni] file
```

Opzioni generali

`-b`	dump ottale
`-d`|`-D`	dump decimale (-d) o decimale esteso (-D)
`-s`|`-S`	dump decimale marcato (-s) o decimale marcato esteso (-S)
`-f`|`-F`	dump in virgola mobile (-f) o virgola mobile estesa (double) (-F)
`-x`|`-X`	dump esadecimale (-x) o esadecimale esteso (-X)
`-c`|`-C`	dump in formato carattere (byte singolo) o carattere esteso (carattere singolo o multi-byte, a seconda dai settaggi locali)
`-v`	modalità verbose

Esempi:

Per dare un'occhiata all'attuale contenuto del seguente file, un elenco di romanzi di Wodehouse Lord Emsworth:

```
Something Fresh [1915]                  Uncle Dynamite [1948]
Leave it to Psmith [1923]               Pigs Have Wings [1952]
Summer Lightning [1929]                 Cocktail Time [1958]
Heavy Weather [1933]                    Service with a Smile [1961]
Blandings Castle and Elsewhere [1935]   Galahad at Blandings [1965]
Uncle Fred in the Springtime [1939]     A Pelican at Blandings [1969]
Full Moon [1947]                        Sunset at Blandings [1977]
```

si può usare:

```
% od -c wodehouse
0000000 S o m e t h i n g   F r e s h
0000020 [ 1 9 1 5 -] \t U n c l e   D y n
0000040 a m i t e   [ 1 9 4 8 -] \n L e a
0000060 v e   i t   t o   P s m i t h
0000100 [ 1 9 2 3 -] \t P i g s   H a v e
0000120   W i n g s   [ 1 9 5 2 -] \n S u
0000140 m m e r   L i g h t n i n g   [
0000160 1 9 2 9 -] \t C o c k t a i l   T
0000200 i m e   [ 1 9 5 8 -] \n H e a v y
0000220   W e a t h e r   [ 1 9 3 3 -] \t
0000240 S e r v i c e   w i t h a   S
0000260 m i l e   [ 1 9 6 1 -] \n B l a n
0000300 d i n g s   C a s t l e   a n d
0000320   E l s e w h e r e   [ 1 9 3 5
0000340 -] \t G a l a h a d   a t   B l a
0000360 n d i n g s   [ 1 9 6 5 -] \n U n
0000400 c l e   F r e d   i n   t h e
0000420 S p r i n g t i m e   [ 1 9 3 9
0000440 -] \t A   P e l i c a n   a t   B
0000460 l a n d i n g s   [ 1 9 6 9 -] \n
0000500 F u l l   M o o n   [ 1 9 4 7 -]
```

```
0000520 \t S u n s e t   a t   B l a n d
0000540 i n g s   [ 1 9 7 7 -] \n
0000554
```

8.3. Connessioni remote

Tabella 8.3. Comandi per connessioni remote

Comando/Sintassi	Cosa fa
`finger [opzioni] user[@nomehost]`	riporta informazioni sugli utenti delle macchine locali e remote
`ftp [opzioni] host`	trasferisce file utilizzando il protocollo di trasferimento di file
`rcp [opzioni] nomehost`	copia file in remoto dalla macchina corrente in un'altra macchina
`rlogin [opzioni] nomehost`	effettua il login in remoto su un'altra macchina
`rsh [opzioni] nomehost`	shell remota da eseguire su un'altra macchina
`telnet [host [porta]]`	comunica con un altro host utilizzando il protocollo telnet

8.3.1. TELNET e FTP - protocollo di login remoto e di trasferimento di file

TELNET e *FTP* sono protocolli del livello applicazione di Internet. Le specifiche dei protocolli FTP e TELNET sono state realizzate da molte organizzazioni, incluso il Centro Nazionale per le Applicazioni di Supercomputer (NCSA), molti altri domini pubblici e organizzazioni collaborative.

I programmi che implementano il protocollo *TELNET* sono usualmente chiamati telnet, ma non sempre. Alcune notevoli eccezioni sono `tn3270` , `WinQVT` e `QWS3271` , che implementano comunque il protocollo TELNET. TELNET viene usato per effettuare il login remoto su un altro computer in Internet.

I programmi che implementano il protocollo *FTP* sono usualmente chiamati ftp, ma esistono anche delle eccezioni. Il programma chiamato `Fetch` , distribuito dalla Dartmouth College, `WS_FTP` , scritto e distribuito da John Junod e `Ftptool` , scritto da Mike Sullivan, sono implementazioni del protocollo FTP con un'interfaccia utente grafica. Esiste una versione migliore di FTP, `ncftp` che ha caratteristiche aggiuntive, scritto da Mike Gleason.

Comunque, l'implementazione del protocollo FTP viene spesso inclusa nei programmi che implementano TELNET, come quello distribuito dalla NCSA. FTP viene usato per trasferire file tra computer su Internet.

rlogin(1) è un servizio di login remoto che è stato in passato un'esclusiva dello Unix BSD 4.3 di Berkeley. Essenzialmente, offre le stesse funzionalità di telnet(1), eccetto che rlogin(1) lascia passare al computer remoto le informazioni dell'ambiente di login dell'utente. Le macchine possono essere configurate per permettere connessioni da fidati host senza richiedere la password dell'utente. Una versione più sicura di questo protocollo è la Sicura SHell SSH, software scritto da Tatu Ylonen e disponibile via: `ftp://ftp.net.ohio-state.edu/pub/security/ssh` .

Da un prompt Unix, questi programmi possono essere invocati digitando il comando (nome del comando) e il nome (Internet) della macchina remota alla quale ci si vuole connettere. Inoltre si possono specificare diverse opzioni per questi comandi, come mostrato di seguito.

Sintassi

```
telnet [opzioni] [host_remoto [porta]]

tn3270 [opzioni] [host_remoto [porta]]

ftp [opzioni] [host_remoto]
```

Opzioni generali

ftp	telnet	Azione
`-d`		abilità la modalità di debugging
	`-d`	come sopra (solamente SVR4)
`-i`		disabilita il prompt interattivo
`-n`		non tenta un auto-login su una connessione
`-v`		modalità verbose
	`-l` *user*	si connette all'host remoto con il nome utente specificato (solamente SVR4)
	`-8`	percorso dati 8-bit (solamente SVR4)

telnet(1) e `tn3270` hanno un'opzione per specificare il numero di porta a cui connettersi sull'host remoto. Per entrambi i comandi, il numero di porta di default è 23, la porta telnet. Altre porte vengono usate per il debugging dei servizi di rete e per ragioni speciali.

Esempi:

```
% telnet oscar.us.ohio-state.edu
```

```
% tn3270 ohstmvsa.acs.ohio-state.edu
```

```
% ftp magnus.acs.ohio-state.edu
```

La macchina remota richiederà di identificarsi tramite login e password. Spesso, le macchine organizzate come archivi di software o di informazioni, permettono connessioni ftp anonime. Ci si collega tramite ftp(1) alla macchina remota e si effettua il login come `anonymous` (il login `ftp` è equivalente su molte macchine) cioè, quando viene richiesto il login si digita `anonymous` (di solito per la password si inserisce il proprio indirizzo email o qualsiasi altra cosa).

Una volta che si è correttamente collegati a un computer remoto tramite telnet(1) e rlogin(1) (e assumendo che sia stata assegnata l'emulazione del terminale) si potrà utilizzare la macchina come al solito.

Una volta che si è correttamente collegati ad un computer remoto tramite ftp(1), si può trasferire un file in quel computer con il comando `put` o prenderlo da quel computer con il comando `get`. La sintassi è la seguente:

```
put nome-file-locale nome-file-remoto
```

```
get nome-file-locale nome-file-remoto
```

Sono disponibili altri comandi per ftp(1), a seconda della specifica implementazione dell'FTP locale e remoto. Il comando `help` visualizza un elenco di comandi disponibili. Il comando `help` può visualizzare lo scopo di un comando specifico. Esempi di validi comandi sono mostrati di seguito:

`help`	mostra un elenco di comandi disponibili
`help mget`	mostra lo scopo del comando `mget` (prendere file multipli)
`pwd`	mostra la corrente directory di lavoro
`ls` o `dir`	elenca il contenuto delle directory
`cd`	cambia directory
`lcd`	cambia la directory locale
`open`	specifica la macchina alla quale si vuole connettersi
`user`	specifica il proprio id di login (nel caso che non venga richiesto direttamente)
`quit`	esce dal programma FTP

8.3.2. finger - restituisce informazioni riguardo gli utenti

Il comando finger(1) mostra il file `.plan` di un utente specifico o riporta chi è attualmente «loggato» su una specifica macchina. L'utente deve permettere i permessi di lettura generale sul file `.plan`.

Sintassi

```
finger [opzioni] [user[@nomehost]]
```

Opzioni generali

`-l`	forza il formato lungo di output
`-m`	ottiene solo lo username, non il nome e cognome reali
`-s`	forza il formato corto di output

Esempi:

```
brigadier:condron [77]> finger workshop@nyssa
Questo è un esempio di file .plan per l'id workshop di nyssa.
Questo id è stato usato in questa settimana da Frank Fiamingo, Linda
DeBula, e Linda Condron, mentre insegnavano ad usare una nuova ↺
versione
di Unix workshop sviluppata per l'UTS.

Sperando di aver insegnato qualcosa.
Frank, Linda, & Linda
```

```
brigadier: condron [77]> finger
Login     Name                TTY     Idle    When        Where

condron   Linda S Condron     p0              Sun 18:13   lcondron-
mac.acs
frank     Frank G. Fiamingo   p1              Mon 16:19   nyssa
```

8.3.3. Comandi remoti

Alcune macchine Unix possono essere collegate tra loro per formare una rete locale (LAN). In questo caso succede spesso che un utente di una macchina possieda validi accessi di login su molte altre macchine della rete locale. Per questo tipo di utente sono disponibili comandi Unix che forniscono una certa praticità nel realizzare certe operazioni comuni. Poichè questi comandi si focalizzano su comunicazioni con host remoti nella rete locale (o in Internet), i nomi dei comandi iniziano con la lettera *r*: rlogin(1), rsh(1) e rcp(1). La possibilità di accesso remoto tramite questi comandi è sostenuta (opzionalmente) attraverso il file `~/.rhosts` di ogni singolo utente e dal file generale di sistema `/etc/hosts.equiv` . Per ragioni di sicurezza questi possono essere limitati ad alcuni host.

Il comando rlogin(1) permette un accesso di login remoto su un altro host della rete locale. rlogin(1) passa informazioni all'host remoto circa l'ambiente locale, incluso il valore della variabile d'ambiente `TERM`.

Il comando rsh(1) fornisce la possibilità di invocare una shell Unix su un host remoto della rete locale con lo scopo di eseguirci comandi di shell. Questa capacità è simile alla

funzione shell escape disponibile generalmente all'interno di un software di sistema Unix come editor ed email.

Il comando rcp(1) fornisce la possibilità di copiare file dall'host locale ad un host remoto della rete locale.

Sintassi

```
rlogin [-l username] host_remoto

rsh [-l username] host_remoto [comando]

rcp [[user1]@host1:]file_sorgente [[user2]@host2:]file_destinazione
```

dove le parti tra parentesi ([]) sono opzionali. rcp(1) non richiede password, quindi si deve avere il permesso per eseguire comandi remoti su una specifica macchina.

Opzioni generali

`-l` *`username`*	si connette all' host remoto con il nome utente specificato (rlogin(1) & rsh(1))

Il file `.rhosts`, se esiste nella directory home dell'utente sull'host remoto, permette l'accesso tramite rlogin(1), rsh(1) e rcp(1) agli host remoti senza richiedere la password per tale account. Il file `.rhosts` contiene un record per ogni coppia host remoto-username con il quale il proprietario del file `.rhosts` desidera connettersi. Ogni record nel file `.rhosts` è della forma:

`host_remoto utente_remoto`

dove l'elenco *`utente_remoto`* è opzionale. Per esempio, se Heather Jones vuole essere in grado di connettersi alla `macchina1` (dove il suo username è `heather`) dalla `macchina2` (dove il suo username è `jones`) o dalla `macchina3` (dove il suo username è `heather`, lo stesso della `macchina1`), lei potrebbe creare un file `.rhosts` nella sua home directory sulla `macchina1`. Il contenuto di questo file potrebbe essere:

```
macchina2 jones
macchina3

-oppure-

macchina2 jones
macchina3 heather
```

Su alcuni sistemi il file `/etc/hosts.equiv` presta lo stesso scopo per tutti gli utenti, al di fuori del super-user. Così un file `/etc/hosts.equiv` contenente:

`macchina_remota`

permette agli utenti provenienti da *macchina_remota* che hanno gli stessi username della macchina corrente, di connettersi a quest'ultima senza la necessita' di inserire la password.

Un file `/etc/hosts.equiv` che contiene:

macchina_remota utente_remoto

permette a *utente_remoto* su *macchina_remota* di connettersi alla macchina corrente allo stesso modo dell'utente locale, tranne che per il super-user.

I file `/etc/hosts.equiv` e `~/.rhosts` vanno usati con cautela.

Le versioni dei programmi rcp(1), rsh(1) e rlogin(1) basati sulla Secure SHell (SSH) sono liberamente disponibili e forniscono molta più sicurezza.

Capitolo 9. Programmazione di shell

9.1. Script di shell

Si possono scrivere programmi di shell creando script contenenti alcuni comandi di shell. La prima linea dello script deve iniziare con #!, che indica al kernel che lo script è direttamente eseguibile. Si fa immediatamente seguire a quel simbolo il nome della shell o del programma da eseguire (gli spazi sono permessi), usando un path name assoluto. Generalmente si possono avere fino a 32 caratteri, forse di più su alcuni sistemi e si può includere qualche opzione. Quindi per inizializzare uno script per la shell Bourne la prima linea dovrà essere:

```
#! /bin/sh
```

e per la shell C:

```
#! /bin/csh -f
```

dove l'opzione `-f` indica che la shell non deve leggere il file `.cshrc`. Alcuni spazi bianchi seguenti il magico simbolo, #!, sono opzionali.

Inoltre si deve specificare che lo script è eseguibile, settando gli opportuni bit sul file con il comando chmod(1), esempio:

```
% chmod +x shell_script
```

All'interno degli script il simbolo # indica un commento da quel punto fino alle fine della linea; #! è un caso speciale se trovato come primo carattere del file.

9.2. Settare i valori dei parametri

I valori di un parametro, ad esempio `param`, sono assegnati così:

Shell Bourne	Shell C
`param=valore`	`set param = valore`

dove *`valore`* è una valida stringa che può essere chiusa tra caratteri di quoting singoli ('*`valore`*') o doppi ("*`valore`*") per permettere alcuni spazi bianchi all'interno del valore della stringa. Quando viene racchiusa con dei caratteri backquote (`` `*valore*` ``) la stringa viene prima valutata dalla shell e viene sostituita con il risultato ottenuto dalla valutazio-

ne. Questo viene spesso usato per eseguire un comando, sostituendo l'output del comando a *valore*, esempio:

```
$ day=`date +%a`
```

```
$ echo $day
Wed
```

Dopo che il valore del parametro è stato assegnato, si accede al valore corrente del parametro usando la notazione `$param` o `${param}`.

9.3. Quoting

Le stringhe possono essere quotate per controllare il modo in cui la shell interpreta alcuni parametri o variabili all'interno della stringa. Per delimitare le stringhe si possono usare i caratteri di quoting singoli (') o doppi ("). I caratteri di quoting doppi definiscono la stringa e permettono la sostituzione di variabile. I carattere di quoting singoli definiscono la stringa ma impediscono la sostituzione di variabile. Un backslash (\) prima di un carattere viene posto per effettuare un escape su di esso, specificando che il sistema deve considerare il carattere letteralmente, senza assegnarli alcun significato speciale. Queste tecniche di quoting possono essere usate per separare una variabile da una stringa fissa. Come esempio si consideri la variabile `var`, a cui è stata assegnata il valore `bat`, e la stringa costante `man`. Se si vuole combinare queste stringhe per ottenere come risultato la stringa `batman` si può sperimentare:

```
$varman
```

ma questo non funzionerà, poichè la shell tenta di valutare una variabile chiamata `varman`, che non esiste. Per ottenere il risultato desiderato si ha la necessità di separare le stringhe tramite quoting o di isolare la variabile con delle parentesi graffe ({}), in questo modo:

`"$var"man`	- quoting sulla variabile
`$var""man`	- separazione di parametri
`$var"man"`	- quoting sulla costante
`$var''man`	- separazione di parametri
`$var'man'`	- quoting sulla costante
`$var\man`	- separazione di parametri
`${var}man`	- si isola la variabile

Queste funzionano tutte poichè ", ', \, { e } non sono validi caratteri per un nome di variabile.

Non si può usare

```
'$var'man
```

```
\$varman
```

poichè impediscono che la sostituzione della variabile prenda posto.

Quando si usano le parentesi graffe, queste devono circondare solamente la variabile, senza includere il $, altrimenti saranno incluse come parte del risultato della stringa, esempio:

```
% echo {$var}man
{bat}man
```

9.4. Variabili

Alcune variabili sono automaticamente inizializzate all'avvio della shell. Queste variabili permettono di riferirsi agli argomenti su linea di comando.

Queste *variabili di shell* sono:

Tabella 9.1. Variabili di shell

Variabile	Uso
`$#`	numero di argomenti su linea di comando
`$-`	opzioni fornite alla shell
`$?`	valore di uscita dell'ultimo comando eseguito
`$$`	numero id del processo corrente
`$!`	numero di processo dell'ultimo comando messo in background
`$n`	argomento su linea di comando, dove *n* varia tra 1 e 9, leggendo da sinistra a destra
`$0`	il nome della shell corrente o del programma corrente
`$*`	tutti gli argomenti su linea di comando ("$1 $2 ... $9")
`$@`	tutti gli argomenti su linea di comando, ciascuno quotato separatamente ("$1" "$2" ... "
`$argv[n]`	seleziona l'*n-esima* parola dalla lista di input
`${argv[n]}`	come sopra
`$#argv`	riporta il numero di parole della lista di input

L'uso di queste variabili può essere illustrato con alcuni semplici script. Per la shell Bourne lo script potrebbe essere:

```
#!/bin/sh
echo "$#:" $#
```

```
echo '$#:' $#
echo '$-:' $-
echo '$?:' $?
echo '$$:' $$
echo '$!:' $!
echo '$3:' $3
echo '$0:' $0
echo '$*:' $*
echo '$@:' $@
```

Quando viene eseguito con alcuni argomenti, mostra i valori delle variabili di shell, esempio:

```
$ ./variables.sh one two three four five
5: 5
$#: 5
$-:
$?: 0
$$: 12417
$!:
$3: three
$0: ./variables.sh
$*: one two three four five
$@: one two three four five
```

Come si può notare, si ha la necessità di usare un carattere di quoting singolo per impedire alla shell di assegnare significati speciali a `$`. Il carattere di quoting doppio, come nella prima struttura `echo`, permette di rimpiazzare il nome della variabile con il suo valore.

Similmente, per le variabili della shell `C` si possono illustrare le sostituzioni di variabili tramite il seguente script:

```
#!/bin/csh -f
echo '$$:' $$
echo '$3:' $3
echo '$0:' $0
echo '$*:' $*
echo '$argv[2]:' $argv[2]
echo '${argv[4]}:' ${argv[4]}
echo '$#argv:' $#argv
```

che quando eseguito con alcuni argomenti mostra il risultato seguente:

```
% ./variables.csh one two three four five

$$: 12419
$3: three
$0: ./variables.csh
$*: one two three four five
$argv[2]: two
${argv[4]}: four
$#argv: 5
```

9.5. Sostituzione di parametri

Si può riferirsi ai parametri in modo astratto e sostituire i loro valori in base a delle condizioni, usando gli operatori definiti qui sotto. Ancora una volta si possono usare le parentesi graffe ({}) per isolare la variabile e il suo operatore.

`$parametro`	sostituisce questa stringa con il valore di *`parametro`*
`${parametro}`	come sopra. Le parentesi sono d'aiuto se non c'è separazione tra questo parametro e una stringa adiacente.
`$parametro=`	setta *`parametro`* a *null*.
`${parametro-default}`	se *`parametro`* non è settato allora si usa *`default`* come valore. *`parametro`* non viene resettato.
`${parametro=default}`	se *`parametro`* non è settato allora lo si setta a *`default`* e si usa il nuovo valore
`${parametro+nuovo_valore}`	se *`parametro`* è settato allora si usa *`nuovo_valore`* altrimenti non si usa nulla. *`parametro`* non viene resettato.
`${parametro?messaggio}`	se *`parametro`* non è settato allora si visualizza il messaggio. Se *`parametro`* è settato allora si usa il valore corrente.

Non ci sono spazi nei precedenti operatori. Se un due punti (:) viene inserito prima di -, =, + o ? allora si effettua prima un test per vedere se il parametro ha un settaggio *non-nullo*.

La shell C ha alcuni modi aggiuntivi per la sostituzione di parametri:

`$lista[n]`	seleziona l'*`n-esima`* parola dalla lista
`${lista[n]}`	come sopra
`$#lista`	riporta il numero di parole in lista
`$?parametro`	ritorna 1 se il parametro è settato, 0 altrimenti
`${?parametro}`	come sopra
`$<`	legge una linea da stdin(4)

Inoltre la shell C definisce l'array `$argv[n]` per contenere gli *n* argomenti della linea di comando e `$#argv` per il numero di argomenti, come specificato in Tabella 9.1.

Per illustrare alcune di queste caratteristiche si userà il seguente script di prova:

```
#!/bin/sh
param0=$0
```

```
test -n "$1" && param1=$1
test -n "$2" && param2=$2
test -n "$3" && param3=$3
echo 0: $param0
echo "1: ${param1-1}: \c" -;echo $param1
echo "2: ${param2=2}: \c" -;echo $param2
echo "3: ${param3+3}: \c" -;echo $param3
```

Inizialmente nello script si verifica con test(1) se la variabile esiste; in tal caso si setta il parametro al suo valore. Dopo si riportano i valori, effettuando le sostituzioni.

Nella prima esecuzione dello script non vengono forniti argomenti:

```
$ ./parameter.sh
0: ./parameter.sh       # trova sempre $0
1: 1:                   # sostituisce  1, ma non assegna questo ↺
valore
2: 2: 2                 # sostituisce 2 e assegna questo valore
3: :                    # non sostituisce
```

In questa seconda esecuzione dello script si forniscono alcuni argomenti:

```
$ ./parameter one two three
0: ./parameter.sh       # trova sempre $0
1: one: one             # non sostituisce, ha già un valore
2: two: two             # non sostituisce, ha già un valore
3: 3: three             # sostituisce 3 ma non assegna questo valore
```

9.6. Here document

Un *here document* è una forma di quoting che permette alle variabili di shell di essere sostituite. È una forma speciale di redirezione che inizia con una linea contenente solamente *<<PAROLA* e finisce con una linea contenete solamente *PAROLA*. Nella shell Bourne si può impedire la sostituzione di shell effettuando un escape su *PAROLA*, mettendo un \ davanti a *PAROLA* sulla linea di redirezione, esempio *<<\PAROLA*, ma non sulla linea finale. Per avere lo stesso effetto con la shell C si mette il \ davanti a *PAROLA* in entrambi i posti.

Gli script che seguono illustrano questo meccanismo:

per la *shell Bourne*:

```
#!/bin/sh
fa=fa
non=""
cat << EOF
Questo here document
$non $fa
sostituzione di variabile
EOF
cat << \EOF
```

e per la *shell C:*

```
#!/bin/csh -f
set fa = fa
set non = ""
cat << EOF
Questo here document
$non $fa
sostituzione di variabile
EOF
cat << \EOF
```

```
Questo here document                    Questo here document
$non $fa                                $non $fa
sostituzione di variabile               sostituzione di variabile
EOF                                     \EOF
```

Entrambi gli output producono:

```
Questo here document
fa
sostituzione di variabile
Questo here document
$non $fa
sostituzione di variabile
```

Nella parte superiore dell'esempio le variabili di shell `$non` e `$fa` sono sostituite. Nella parte inferiore queste variabili vengono trattate come delle semplici stringhe di testo senza effettuare la sostituzione.

9.7. Input interattivo

Gli script di shell possono accettare input interattivo per inizializzare parametri all'interno dello script stesso.

9.7.1. Sh

sh(1) utilizza il comando built-in `read` per leggere una linea di input, esempio:

```
read param
```

Questo può essere illustrato con un semplice script:

```
#!/bin/sh
echo "Inserisci una frase \c"   # /bin/echo che richiede "\c" per
                                # impedire un newline
read param
echo param=$param
```

Quando questo script viene eseguito, viene richiesto l'input, che viene poi mostrato nel risultato:

```
$ ./read.sh
Inserisci una frase  hello frank   # E' stato digitato hello frank ↺
<return>
param=hello frank
```

9.7.2. Csh

csh(1) usa il simbolo `$<` per leggere una linea da stdin(4), esempio:

```
set param = $<
```

Gli spazi bianchi intorno al segno di uguale sono importanti. Il seguente script illustra come usarlo:

```
#!/bin/csh -f
  echo -n "Inserisci una frase"    # Questo echo built-in richiede -n
                                    # per impedire un newline
  set param = $<
  echo param=$param
```

Quindi chiede l'input e lo mostra nel risultato:

```
% ./read.csh
Inserisci una frase  hello frank  # E' stato digitato hello frank ↺
<return>
param=hello frank
```

9.8. Funzioni

La shell Bourne permette di definire funzioni. Queste sono molto simili agli alias della shell C, ma permettono più flessibilità. Una funzione ha la forma:

```
funzione () { comando; }
```

dove lo spazio dopo { e il punto e virgola (;) sono obbligatori; il punto e virgola può essere omesso facendo precedere a } un newline. Spazi e newline aggiuntivi sono permessi. Alcuni esempi di funzioni possono essere visti nel semplice file `.profile` discusso nei primi capitoli, dove si avevano delle funzioni per `ls` e `ll`:

```
ls() { /bin/ls -sbF "$@";}

ll() { ls -al "$@";}
```

La prima funzione ridefinisce ls(1) affinchè le opzioni `-sbF` siano sempre fornite dal comando standard `/bin/ls` e in modo da agire in base all'input fornito, `$@`. La seconda di queste funzioni prende il valore corrente di `ls` (la funzione precedente) e aggiunge le opzioni `-al`.

Le funzioni sono molto utili negli script di shell. Il seguente script è una versione semplificata di uno script utilizzato per effettuare automaticamente il backup su nastro delle partizioni di sistema.

```
#!/bin/sh
# Script cron per un completo backup del sistema
HOST=`/bin/uname -n`
admin=frank
Mt=/bin/mt
Dump=/usr/sbin/ufsdump
Mail=/bin/mailx
device=/dev/rmt/0n
```

```
Rewind="$Mt -f $device rewind"
Offline="$Mt -f $device rewoffl"
# Failure - exit
failure () {
  $Mail -s "Backup fallito - $HOST" $admin << EOF_failure
  $HOST
  Script cron backup fallito. A quanto pare non c'è il nastro nel ↺
dispositivo.
  EOF_failure
  exit 1
 }
# Dump failure - exit
dumpfail () {
  $Mail -s "Backup fallito - $HOST" $admin << EOF_dumpfail
  $HOST
  Script cron backup fallito. A quanto pare non c'è il nastro nel ↺
dispositivo.
  EOF_dumpfail
  exit 1
 }
# Success
success () {
  $Mail -s "Backup completato con successo - $HOST" $admin << ↺
EOF_success
  $HOST
  Script cron backup apparentemente riuscito. Il file /etc/
dumpdates è:
  `/bin/cat /etc/dumpdates`
  EOF_success
  }
# Conferma nastro nel device
$Rewind || failure
$Dump 0uf $device / || dumpfail
$Dump 0uf $device /usr || dumpfail
$Dump 0uf $device /home || dumpfail
$Dump 0uf $device /var || dumpfail
($Dump 0uf $device /var/spool/mail || dumpfail) && success
$Offline
```

Questo script illustra alcuni argomenti che sono stati trattati in questo documento. Lo script inizia settando i valori di alcuni parametri. `HOST` viene inizializzato dall'output di un comando, `admin` è l'amministratore di sistema, `Mt`, `Dump` e `Mail` sono nomi di programmi, `device` è il dispositivo speciale usato per accedere al nastro, `Rewind` e `Offline` contengono i comandi rispettivamente per riavvolgere e scaricare il nastro usando il riferimento `Mt` e le necessarie opzioni. Vengono definite tre funzioni: `failure`, `dumpfail` e `success`. Tutte le funzioni in questo script utilizzano la forma *here document* per realizzare il contenuto della funzione stessa. Si introducono ora gli operatori logici *OR* (*||*) e *AND* (*&&*); ciascuno è posizionato tra una coppia di comandi. Per l'operatore *OR*, il secondo comando viene eseguito solamente se il primo comando non è stato completato con successo. Per l'operatore *AND*, il secondo comando viene eseguito solamente se il primo comando è stato completato con successo.

Lo scopo principale dello script è realizzare i comandi `Dump`, ad esempio copiando i file system specificati. Prima si tenta di eseguire il riavvolgimento del nastro. Se questo fallisce, `|| failure`, si esegue la funzione `failure` e si esce dal programma. Se invece questo ha successo si procede con il backup a turno di ogni partizione, ogni volta verificando che questa operazione sia completamente riuscita (`|| dumpfail`). Se questa operazione non viene eseguita completamente con successo si esegue la procedura `dumpfail` e si esce. Se l'ultimo backup ha successo si procede con la funzione `success` (`(...) && success`). In fine si riavvolge il nastro e lo si manda fuori linea affinchè altri utenti non possano accidentalmente scriverci sopra.

9.9. Comandi di controllo

9.9.1. Condizionale if

L'espressione condizionale `if` è disponibile in entrambe le shell, ma con una diversa sintassi.

9.9.1.1. Sh

```
if condizione1
then
        lista di comandi se condizione1  è vera (true)
[elif condizione2
        then lista di comandi se condizione2  è vera (true)]
[else
        lista di comandi se condizione1  è falsa (false)]
fi
```

Le condizioni sono sottoposte usualmente al comando test(1) o `[]` (Vedere la sezione 9.9.6). L'`if` e `then` devono essere separati con un newline o un punto e virgola (;).

```
#!/bin/sh
if [ $# -ge 2 -]
then
  echo $2
elif [ $# -eq 1 -]; then
  echo $1
else
  echo Nessun input
fi
```

Sono richiesti degli spazi nel formato della condizione di test(1), uno dopo `[` e uno prima di `]`. Questo script potrebbe comportarsi in modo differente a seconda che ci siano zero, uno o più argomenti su linea di comando. Iniziando con nessun argomento:

```
$ ./if.sh
Nessun input
```

Ora con un argomento:

```
$ ./if.sh one
one
```

E ora con due argomenti:

```
$ ./if.sh one two
two
```

9.9.1.2. Csh

```
if (condizione ) comando
   -oppure-
if (condizione1 ) then
        lista di comandi se condizione1  è vera (true)
[else if (condizione2 ) then
        lista di comandi se condizione2  è vera (true)]
[else
        lista di comandi se condizione1  è falsa (false)]
endif
```

L'`if` e `then` devono stare sulla stessa linea.

```
#!/bin/csh -f
if ( $#argv >= 2 ) then
  echo $2
else if ( $#argv == 1 ) then
  echo $1
else
  echo Nessun input
endif
```

Di nuovo, questo script potrebbe comportarsi in modo differente a seconda che ci siano zero, uno o più argomenti su linea di comando. Iniziando con nessun argomento:

```
% ./if.csh
Nessun input
```

Ora con un argomento:

```
% ./if.csh one
one
```

E ora con due argomenti:

```
% ./if.csh one two
two
```

9.9.2. Condizionale switch e case

Per scegliere tra una serie di valori di tipo stringa relativi a un parametro si usa `case` nella shell Bourne e `switch` nella shell C.

9.9.2.1. Sh

```
case parametro  in
        schema1 [|schema1a ]) lista1 di comandi ;;
        schema2 ) lista2 di comandi
                  lista2a di comandi ;;
        schema3 ) lista3 di comandi ;;
        *) -;;
esac
```

Si possono usare validi nomi di file meta-caratteri all'interno dello schema per il confronto. I ;; concludono ogni scelta e possono essere sulla stessa linea della scelta o a seguito di un newline, dopo l'ultimo comando per la scelta. Schemi alternativi per la scelta di un particolare caso sono separati da una barra verticale |, come nel primo schema dell'esempio precedente. I simboli wild card ,? per indicare un generico carattere e * per far corrispondere alcuni caratteri, possono essere usati singolarmente o in modo adiacente per completare stringhe.

Questo semplice esempio illustra come usare l'espressione condizionale `case`.

```
#!/bin/sh
case $1 in
        aa|ab) echo A
        -;;
        b?)    echo "B \c"
               echo $1;;
        c*)    echo C;;
        *)     echo D;;
esac
```

Quindi quando si esegue lo script con l'argomento posto sulla colonna di sinistra, lo script risponde come sulla colonna di destra:

aa	A
ab	A
ac	D
bb	B bb
bbb	D
c	C
cc	C
fff	D

9.9.2.2. Csh

```
switch (parametro )
case schema1 :
        lista1 di comandi
```

```
        [breaksw]
case schema2:
        lista2 di comandi
        [breaksw]
default:
        lista di comandi per il comportamento di default
        [breaksw]
endsw
```

`breaksw` è opzionale e può essere usato per interrompere lo switch dopo che si è verificata una corrispondenza del valore di tipo stringa del parametro confrontato. `Switch` non accetta | nella lista degli schemi, ma permette di unire insieme diverse strutture `case` per fornire un simile risultato. Il seguente script di shell `C` ha lo stesso comportamento dell'esempio precedente, riferito al `case` della shell Bourne.

```
#!/bin/csh -f
switch ($1)
        case aa:
        case ab:
                echo A
                breaksw
        case b?:
                echo -n "B "
                echo $1
                breaksw
        case c*:
                echo C
                breaksw
        default:
                echo D
endsw
```

9.9.3. for e foreach

Per effettuare un ciclo tra una lista di valori di tipo stringa si possono usare i comandi `for` e `foreach`.

9.9.3.1. Sh

```
for variabile [in lista_di_valori ]
do
        lista di comandi
done
```

La *`lista_di_valori`* è opzionale, presupponendo `$@` se nulla viene specificato. Ogni valore in questa lista viene sostituito sequenzialmente in *`variabile`* fino a quando la lista risulta vuota. Possono essere usati wild card, che vengono applicati ai nomi dei file nella directory corrente. Di seguito si illustra il ciclo `for` che copia tutti i file che finiscono con `.old` negli stessi nomi che finiscono però con `.new`. In questi esempi l'utility basename(1) estrae la parte base del nome affinchè si possa modificarne l'estensione.

```
#!/bin/sh
for file in *.old
do
        newf=`basename $file .old`
        cp $file $newf.new
done
```

9.9.3.2. Csh

```
foreach variabile (lista_di_valori )
        lista di comandi
end
```

L'equivalente script in shell C per copiare tutti i file con estensione `.old` negli stessi file con estensione `.new` è:

```
#!/bin/csh -f
foreach file (*.old)
        set newf = `basename $file .old`
        cp $file $newf.new
end
```

9.9.4. while

Il comando `while` permette di effettuare il ciclo sempre che la condizione sia vera.

9.9.4.1. Sh

```
while condizione
do
        lista di comandi
        [break]
        [continue]
done
```

Un semplice script per illustrare il ciclo `while` è:

```
#!/bin/sh
while [ $# -gt 0 -]
do
        echo $1
        shift
done
```

Questo script prende la lista degli argomenti, ne visualizza il primo, quindi effettua uno shift nella lista verso sinistra, perdendo il primo elemento originale. Il ciclo viene ripetuto fino a quando tutti gli argomenti sono stati spostati fuori dalla lista.

```
$ ./while.sh one two three
one
two
```

```
three
```

9.9.4.2. Csh

```
while (condizione )
        lista di comandi
        [break]
        [continue]
end
```

Se si vuole che la condizione sia sempre vera si specifica 1 all'interno del test condizionale.

Lo script di shell C equivalente a quello precedente è:

```
#!/bin/csh -f
while ($#argv != 0 )
        echo $argv[1]
        shift
end
```

9.9.5. until

Questo costrutto di ciclo è solamente disponibile per la shell Bourne.

```
until condizione
do
        lista di comandi se la condizione è falsa
done
```

La condizione viene verificata all'inizio di ogni ciclo e il ciclo termina quando la condizione è vera.

Uno script equivalente all'esempio del `while` precedente è:

```
#!/bin/sh
until [ $# -le 0 -]
do
        echo $1
        shift
done
```

Si noti che qui si verifica per *minore o uguale*, piuttosto che per *maggiore*, poichè il ciclo `until` viene abilitato da una condizione *falsa*.

Sia il ciclo `until` che il `while` sono solamente eseguiti se la condizione è soddisfatta. La condizione viene valutata prima dell'esecuzione dei comandi.

9.9.6. test

Le espressioni condizionali vengono valutate per valori *veri* o *falsi*. Questo, di solito, viene realizzato con test(1) o equivalentemente con i suoi operatori `[]`. Se la condizione viene

valutata vera, viene settato uno stato di uscita zero (*TRUE*), altrimenti viene settato uno stato di uscita non-zero (*FALSE*). Se non ci sono argomenti viene settato uno stato di uscita non-zero. Gli operatori utilizzati nelle espressioni condizionali della shell Bourne sono mostrati qui sotto.

Per i *nomi di file* le opzioni per test(1) sono date con la sintassi seguente:

`-opzione filename`

Le opzioni di test(1) disponibili per i *file* includono:

`-r`	vero se il file esiste ed è leggibile
`-w`	vero se il file esiste ed è scrivibile
`-x`	vero se il file esiste ed è eseguibile
`-f`	vero se il file esiste ed è un file regolare (o per csh(1) esiste e non è una directory)
`-d`	vero se il file esiste ed è una directory
`-h` o `-L`	vero se il file esiste ed è un link simbolico
`-c`	vero se il file esiste ed è un file speciale a caratteri (ad esempio un dispositivo al quale si accede un carattere alla volta)
`-b`	vero se il file esiste ed è un file speciale a blocchi (ad esempio un dispositivo al quale si accede in blocchi di dati)
`-p`	vero se il file esiste ed è un file pipe (fifo)
`-u`	vero se il file esiste ed è setuid (ad esempio ha il bit set-user-id settato a s o S nel terzo bit)
`-g`	vero se il file esiste ed è setgid (ad esempio ha il bit set-group-id settato a s o S nel sesto bit)
`-k`	vero se il file esiste e ha lo sticky bit settato (una t nel nono bit)
`-s`	vero se il file esiste ed ha una dimensione maggiore di zero

C'è un test per i *descrittori di file*:

`-t [descrittore_file ]`	vero se l'aperto descrittore del file specificato (1, stdout(4), di default) è associato ad un terminale

Ci sono test per le *stringhe*:

`-z stringa`	vero se la lunghezza della stringa è zero
`-n stringa`	vero se la lunghezza della stringa non è zero
`stringa1 = stringa2`	vero se `stringa1` è identica a `stringa2`

stringa1 != *stringa2*	vero se *stringa1* non è identica a *stringa2*
stringa	vero se la stringa non è nulla

Ci sono dei confronti per gli interi:

n1 -eq *n2*	vero se gli interi *n1* e *n2* sono uguali
n1 -ne *n2*	vero se gli interi *n1* e *n2* non sono uguali
n1 -gt *n2*	vero se l'intero *n1* è maggiore dell'intero *n2*
n1 -ge *n2*	vero se l'intero *n1* è maggiore o uguale dell'intero *n2*
n1 -lt *n2*	vero se l'intero *n1* è minore dell'intero *n2*
n1 -le *n2*	vero se l'intero *n1* è minore o uguale dell'intero *n2*

Sono disponibili i seguenti *operatori logici*:

!	negazione (unaria)
-a	and (binario)
-o	or (binario)
()	le espressioni all'interno di ( ) vengono raggruppate insieme. Può essere necessario quotare le parentesi ( ) per impedire alla shell di interpretarle.

9.9.7. Operatori relazionali e logici della shell C

La shell C possiede un suo set di operatori logici e relazionali built-in. In ordine decrescente di priorità questi sono:

(...)	raggruppa espressioni con ( )
~	inverso (il suo complemento)
!	negazione logica
*, /, %	moltiplicazione, divisione, modulo
+, -	addizione, sottrazione
<<, >>	shift a sinistra di bit, shift a destra di bit
<=	minore o uguale
>=	maggiore o uguale
<	minore
>	maggiore
= =	uguale
!=	non uguale

`=~`	uguale a stringa
`!~`	non uguale a stringa
`&`	AND bit
`^`	XOR bit (or esclusivo)
`|`	OR bit
`&&`	AND logico
`||`	OR logico
`{comando}`	vero (1) se il comando termina con uno stato di uscita 0, falso (0) altrimenti.

Inoltre la shell C permette richieste sul tipo e sui permessi dei file con gli operatori seguenti:

`-r`	ritorna vero (1) se il file esiste ed è leggibile, altrimenti ritorna falso (0)
`-w`	vero se il file esiste ed è scrivibile
`-x`	vero se il file esiste ed è eseguibile
`-f`	vero se il file esiste e non è una directory
`-d`	vero se il file esiste ed è una directory
`-e`	vero se il file esiste
`-o`	vero se l'utente corrente è il proprietario del file
`-z`	vero se il file ha una lunghezza zero (file vuoto)

Capitolo 10. Editor

Esistono numerose utility per la manipolazione del testo in Unix, come è stato notato attraverso questo documento (esempio ed(1), ex(1), sed(1), awk(1), la famiglia grep e la famiglia roff). Tra gli editor, l'editor visuale (o a schermo pieno) standard su Unix è vi. Questa applicazione comprende un super-set, per così dire, di caratteristiche di ed(1) e di ex(1) (gli editor a linea di Unix).

vi(1) è un editor modale. Questo significa che ha modalità specifiche che permettono l'inserimento del testo, la cancellazione del testo e l'inserimento dei comandi. Si può lasciare la modalità di inserimento premendo il tasto escape. In questo modo ci si porta nella modalità comando. L'editor di linea ex(1) è incorporato in vi(1). Si può passare dalla modalità a schermo pieno a quella a linea di comando (e viceversa) quando si desidera. In modalità *vi* premere Q per andare in modalità *ex*. In modaltà *ex*, al prompt : digitare `vi` per ritornare in modalità *vi*. Inoltre è disponibile una modalità di vi(1) in sola lettura, che può essere invocata con view(1).

Un altro editor comune nei sistemi Unix, specialmente nei college e negli ambienti universitari, è emacs (che sta per «editing macros»). Mentre vi(1) è in generale compreso nel sistema operativo Unix, emacs(1) usualmente non lo è. Emacs viene distribuito dalla Free Software Foundation ed è discutibilmente il più potente editor disponibile per Unix. Emacs è un software di sistema molto grande ed è un grossa risorsa di sistema per un utente di computer.

La Free Software Foundation e il progetto GNU (del quale emacs fa parte) sono stati fondati da Richard Stallman e dai suoi soci, i quali credono (come specificato nel manifesto GNU) che condividere il software sia «l'atto fondamentale di amicizia tra programmatori». La loro General Public License garantisce il diritto d'uso, di modifica e di distribuzione di emacs (incluso il suo codice sorgente) ed è stata progettata specificatamente per impedire a qualcuno di prendersi un profitto finanziario da emacs o da altri software conseguiti attraverso la Free Software Foundation. Molti dei loro software, incluso emacs, sono disponibili via `ftp://ftp.gnu.org/pub/gnu/` e `http://www.gnu.org/` .

Sia vi(1) che emacs(1) permettono di creare file di inizializzazione che possono contenere macro per controllare i settaggi e le funzioni degli editor.

10.1. Configurare la propria sessione vi

Per configurare l'ambiente di vi(1) certe opzioni possono essere settate con il comando di linea dell'editor `:set` durante una sessione di editing. In alternativa, le opzioni usate di frequente possono essere automaticamente settate quando viene invocato vi(1), attraverso il file `.exrc`. Inoltre, questo file può contenere macro per mappare battute di tasti in funzioni usando la funzione `map`. All'interno di vi(1) queste macro possono essere definite con il comando `:map`. I caratteri di controllo possono essere inseriti digitando prima

Ctrl+V, quindi il carattere di controllo desiderato. Alcune opzioni disponibili in vi(1) sono mostrate qui sotto. Alcuni sistemi Unix non accettano certe di queste opzioni.

`:set all`	visualizza tutti i settaggi delle opzioni
`:set ignorecase`	ignora il maiuscolo e minuscolo di un carattere in una ricerca
`:set list`	visualizza tab e return
`:set nolist`	mette off l'opzione `list`
`:set number`	visualizza i numeri di linea
`:set nonumber`	disattiva i numeri di linea
`:set showmode`	visualizza l'indicazione che la modalità di inserimento è on
`:set noshowmode`	mette off l'opzione `showmode`
`:set wrapmargin=n`	mette on la modalità word-wrap a *n* spazi dal margine destro
`:set wrapmargin=0`	mette off l'opzione `wrapmargin`
`:set warn`	visualizza l'avvertimento «Ultima modifica non registrata»
`:set nowarn`	mette off l'avvertimento di scrittura `warn`

Segue un esempio di file `.exrc`:

```
set wrapmargin=10
set number
set list
set warn
set ignorecase
map K {!}fmt -80        # riformatta questo paragrafo, {!}, usando ↺
fmt a
                        # 80 caratteri per linea
map ^Z :!spell          # invoca spell, :!, per verificare lo ↺
splelling di
                        # una parola (ritorna a vi con Ctrl+D)
```

10.2. Configurare la propria sessione emacs

Configurare l'ambiente di emacs(1) equivale a creare chiamate a funzioni LISP. emacs(1) è infinitamente personalizzabile tramite variabili *emacs*, funzioni built-in e attraverso la programmazione Emacs LISP. I settaggi possono essere specificati dal minibuffer (o da linea di comando) durante una sessione *emacs*. Alternativamente, i settaggi usati di frequente possono essere attivati automaticamente quando viene invocato emacs(1), usan-

do il file `.emacs`. Benchè una discussione del Emacs LISP vada oltre lo scopo di questo documento, seguono alcuni esempi di configurazioni per emacs(1).

Per settare o verificare variabili *emacs* o per usare le sue funzioni built-in, si usa il tasto escape (*Meta* è come emacs(1) si riferisce a questo) seguito dalla lettera x, quindi la variabile o la funzione e i suoi argomenti.

`M, x what-line`	quale linea è sul cursore?
`M, x auto-fill-mode`	mette on word-wrap
`M, x auto-fill-mode`	mette off word-wrap
`M, x set-variable <return> fill-column <return> 45`	setta la lunghezza di linea a 45 caratteri
`M, x set-variable <return> auto-save-interval <return> 300`	salva automaticamente il file ogni 300 battute di tasti
`M, x goto-line <return> 16`	muove il cursore alla linea 16
`M, x help-for-help`	invoca l'help di emacs quando Ctrl+h è stato mappato al tasto backspace

Segue un esempio del file `.emacs`:

```
(message "Loading ~/.emacs...")
; I commenti iniziano con un punto e virgola e continuano fino alla ↺
fine della linea.
(setq text-mode-hook 'turn-on-auto-fill) -;mette on word-wrap
(setq fill-column 45)            -;lunghezza di linea pari a 45 ↺
caratteri
(setq auto-save-interval 300)        -;salva il file ogni 300 ↺
battute di tasti
; Costruisce (o mappa) la funzione di rubout (Ctrl+h) nel tasto ↺
backspace
(global-set-key "\C-h" 'backward-delete-char-untabify)
; Costruisce la funzione help emacs per la sequenza di battitura
 "Ctrl+x ?"
(global-set-key "\C-x?" 'help-for-help)
; Per saltare alla linea 16, digitare  M, #<return>16
(global-set-key "\M-#" 'goto-line)
; Per sapere su che linea si è, digitare M, n
(global-set-key "\M-n" 'what-line)
(message "~/.emacs loaded.")
(message "")
```

10.3. Veloce guida per vi

Tutti i comandi in vi(1) sono preceduti dalla pressione del tasto escape. Ogni volta che si deve intraprendere un nuovo comando si deve utilizzare il tasto di escape. Diversamente da dove indicato, vi(1) è case sensitive (sensibile alla differenza minuscolo e maiuscolo).

Comandi movimento cursore:

(*n*) indica un numero ed è opzionale

`(n)h`	(*n*) spazi a sinistra
`(n)j`	(*n*) spazi giù
`(n)k`	(*n*) spazi su
`(n)l`	(*n*) spazi a destra

(Generalmente funzionano anche i tasti freccia)

`Ctrl F`	avanti di una schermata
`Ctrl B`	indietro di una schermata
`Ctrl D`	giù di mezza schermata
`Ctrl U`	su di mezza schermata

(Ctrl indica il tasto control; il case sensitive non è importante)

`H`	all'inizio della linea superiore della schermata
`M`	all'inizio della linea mediana della schermata
`L`	all'inizio dell'ultima linea della schermata
`G`	all'inizio dell'ultima linea del file
`(n)G`	all'inizio della linea (*n*)
`0`	(zero) all'inizio della linea
`$`	alla fine della linea
`(n)w`	avanti (*n*) parole
`(n)b`	indietro (*n*) parole
`e`	fine della parola

Inserimento testo:

`i`	inserimento testo prima del cursore
`a`	aggiunta testo dopo il cursore (non sovrascrive altro testo)

`I`	inserimento testo all'inizio della linea
`A`	aggiunta testo alla fine della linea
`r`	sostituisce il carattere posto sotto il cursore con il prossimo carattere digitato
`R`	sovrascrive i caratteri fino alla fine della linea (o fino a quando il tasto escape viene digitato per cambiare comando)
`o`	(alpha o) inserisce una nuova linea dopo la linea corrente per inserire del testo
`O`	(alpha O) inserisce una nuova linea prima della linea corrente per inserire del testo

Cancellazione testo:

`dd`	cancella la linea corrente
`(n)dd`	cancella (*n*) linee
`(n)dw`	cancella (*n*) parole
`D`	cancella dal cursore fino alla fine della linea
`x`	cancella il carattere corrente
`(n)x`	cancella (*n*) caratteri
`X`	cancella il carattere precedente

Comandi di modifica:

`(n)cc`	modifica (*n*) caratteri sulla linea fino alla fine della linea (o fino a quando viene digitato il tasto escape)
`cw`	modifica i caratteri di una parola fino alla fine della parola (o fino a quando viene digitato il tasto escape)
`(n)cw`	modifica i caratteri delle prossime (*n*) parole
`c$`	modifica il testo alla fine della linea
`ct(x)`	modifica il testo alla lettera (*x*)
`C`	modifica il testo rimanente sulla linea corrente (fino a quando viene digitato il tasto escape)
`~`	modifica il minuscolo/maiuscolo del carattere corrente
`J`	unisce la linea corrente a quella successiva
`u`	annulla l'ultimo comando realizzato sulla linea corrente
`.`	ripete l'ultima modifica

`s`	sostituisce il carattere corrente con il testo digitato
`S`	sostituisce la linea corrente con il testo digitato
`:s`	sostituisce vecchie parole con nuove parole `:<linee considerate> s/vecchio/nuovo/g`
`&`	ripete l'ultimo comando di sostituzione (`:s`)
`(n)yy`	«strappa» (*n*) linee dal buffer
`y(n)w`	«strappa» (*n*) parole dal buffer
`p`	inserisce il testo eliminato o «strappato» dopo il cursore
`P`	inserisce il testo eliminato o «strappato» prima del cursore

Manipolazione file:

`:w (file)`	scrive i cambiamenti nel file specificato (file corrente di default)
`:wq`	scrive i cambiamenti nel file corrente e conclude la sessione di editing
`:w! (file)`	sovrascrive il file (file corrente di default)
`:q`	esce dalla sessione di editing se non sono stati creati cambiamenti
`:q!`	esce dalla sessione di editing e scarta eventuali cambiamenti non salvati
`:n`	edita il prossimo file nella lista dell'argomento
`:f (nome)`	modifica il nome del file corrente in quello specificato
`:r (file)`	legge il contenuto del file specificato all'interno del corrente editing e alla corrente posizione del cursore (inserisce un file)
`:!(comando)`	escape di shell
`:r!(comando)`	inserisce il risultato del comando di shell specificato nella posizione corrente
ZZ	scrive i cambiamenti nel file corrente ed esce

10.4. Veloce guida per emacs

I comandi di emacs(1) sono realizzati sia tramite la simultanea pressione del tasto control (indicato da Ctrl+), sia attraverso il primo colpo del tasto di escape (indicato da M,).

Comandi essenziali

`Ctrl h`	help
`Ctrl x , u`	undo (annulla operazione precedente)
`Ctrl x , Ctrl g`	esce dalla operazione o comando corrente
`Ctrl x , Ctrl s`	salva il file
`Ctrl x , Ctrl c`	chiude emacs(1)

Movimenti cursore

`Ctrl f`	avanti di un carattere
`Ctrl b`	indietro di un carattere
`Ctrl p`	linea precedente
`Ctrl n`	linea successiva
`Ctrl a`	inizio linea
`Ctrl e`	fine linea
`Ctrl l`	al centro della corrente linea sullo schermo
`Ctrl v`	sfoglia avanti
`M, v`	sfoglia indietro
`M, f`	avanti di una parola
`M, b`	indietro di una parola
`M, a`	inizio periodo
`M, e`	fine periodo
`M, {`	inizio paragrafo
`M, }`	fine paragrafo
`M, <`	inizio buffer
`M, >`	fine buffer

Altre funzioni importanti

`M, (n)`	ripete il prossimo comando (*n*) volte
`Ctrl d`	cancella un carattere
`M, d`	cancella una parola
`Ctrl k`	elimina linea
`M, k`	elimina periodo
`Ctrl s`	ricerca in avanti

`Ctrl r`	ricerca al rovescio
`M, %`	sostituzione di query
`M, c`	capitalizza parola
`M, u`	parola in lettere maiuscole
`M, l`	parola in lettere minuscole
`Ctrl t`	commuta caratteri
`M, t`	commuta parole
`Ctrl @`	marca l'inizio di una regione
`Ctrl w`	taglia/cancella tutto dalla marcatura al punto
`Ctrl y`	incolla il testo eliminato o «strappato» all'interno della locazione corrente
`M, q`	riformatta il paragrafo
`M, g`	riformatta ogni paragrafo nella regione

`M, x auto-fill-mode`	mette on word wrap
`M, x set-variable <return> fill-column <return> 45`	setta la lunghezza di linea a 45 caratteri
`M, x goto-line <return> 16`	muove il cursore alla

	li- nea 16
`M, w`	copia la regione marcata
`Ctrl x , Ctrl f`	cerca un file e lo legge
`Ctrl x , Ctrl v`	cerca e legge un file alternativo
`Ctrl x , i`	inserisce un file alla posizione corrente
`Ctrl x , Ctrl s`	salva il file
`Ctrl x , Ctrl w`	scrive il buffer in un file differente
`Ctrl x , Ctrl c`	esce da emacs(1) e chiede di salvare

Capitolo 11. Riassunto dei comandi Unix

11.1. Comandi Unix

Nella tabella che segue vengono riassunti i comandi maggiormente usati su un sistema Unix. In questa tabella, come in generale avviene, per molti comandi Unix, *`file`* può essere un nome di file, una lista di nomi di file o input/output che potrebbe essere rediretto per o dal comando.

Tabella 11.1. Comandi Unix

Comando/Sintassi	Cosa fa
`awk/nawk [opzioni] file`	esamina schemi in un file e processa i risultati
`cat [opzioni] file`	concatena (lista) un file
`cd [directory]`	cambia directory
`chgrp [opzioni] gruppo file`	cambia il gruppo di appartenenza di un file
`chmod [opzioni] file`	cambia i permessi di accesso a file o directory
`chown [opzioni] proprietario file`	cambia il proprietario di un file; può essere solo fatto dal super-user
`chsh (passwd -e/-s) username login_shell`	cambia la shell di login dell'utente

Comando/Sintassi	Cosa fa
	(spesso solamente attraverso il super-user)
`cmp [opzioni] file1 file2`	confronta due file e mostra dove avvengono le differenze (file di testo e file binari)
`compress [opzioni] file`	comprime il file specificato e lo salva con l'estensione `.Z`
`cp [opzioni] file1 file2`	copia `file1` in `file2`; `file2` non dovrebbe già esistere. Questo comando crea o sovrascrive `file2`
`cut (opzioni) [file]`	taglia specifici campi/caratteri dalle linee del file specificato
`date [opzioni]`	riporta data e ora corrente
`dd [if=infile] [of=outfile] [operando=valore]`	copia un file, converte tra ASCII e EBCDIC o scam-

Comando/Sintassi	Cosa fa
	bia l'ordine di byte, come specificato
`diff [opzioni] file1 file2`	confronta i due file e visualizza le differenze (solamente file di testo)
`df [opzioni] [risorsa]`	riporta il sommario dei blocchi del disco e degli inode liberi e usati
`du [opzioni] [directory o file]`	riporta lo spazio di disco usato
`echo [stringa di testo]`	riporta la stringa di testo specificata in standard output
`ed` o `ex [opzioni] file`	editor a linea di comando
`emacs [opzioni] file`	editor a schermo-pieno
`expr argomenti`	valuta gli argomenti. Usato per l'aritmetica, ecc. in shell
`file [opzioni] file`	classifica il tipo di file
`find directory [opzioni] [azioni]`	cerca file basandosi sul

Comando/Sintassi	Cosa fa
	tipo o su uno schema
`finger [opzioni] user[@nomehost]`	riporta informazioni circa gli utenti di macchine locali e remote
`ftp [opzioni] host`	trasferisce file utilizzando il protocollo di trasferimento di file (FTP)
`grep [opzioni] 'stringa di ricerca' argomento`	ricerca nell'argomento (in questo caso probabilmente un file) tutte le occorrenze della stringa di ricerca specificata e le visualizza
`egrep [opzioni] 'stringa di ricerca' argomento`	
`fgrep [opzioni] 'stringa di ricerca' argomento`	
`gzip [opzioni] file`	comprime o decomprime un file. I file compressi vengo memorizzati con l'estensione `.gz`
`gunzip [opzioni] file`	
`zcat [opzioni] file`	
`head [-numero] file`	mostra le prime 10 (o numero di) linee di un file

Comando/Sintassi	Cosa fa
`hostname`	mostra o setta (solamente super-user) il nome della macchina attuale
`kill [opzioni] [-SEGNALE] [pid#] [%job]`	manda un segnale al processo specificato dal numero di processo id (*`pid#`*) o dal numero di controllo del job (`%`*n*). Il segnale di default termina il processo
`ln [opzioni] sorgente destinazione`	crea un collegamento di nome `destinazione` a `sorgente`
`lpq [opzioni]`	mostra lo stato dei job di stampa
`lpstat [opzioni]`	
`lpr [opzioni] file`	stampa sulla stampante specificata
`lp [opzioni] file`	
`lprm [opzioni]`	rimuove un job di stampa dalla coda di stampa
`cancel [opzioni]`	
`ls [opzioni] [directory o file]`	elenca il contenuto della directory specifica-

Comando/Sintassi	Cosa fa
	ta o i permessi del file specificato
`mail [opzioni] [user]`	semplice utility per la posta elettronica disponibile su un sistema Unix. Si digita un punto come primo carattere su una nuova linea per trasmettere il messaggio, un punto interrogativo per richiamare l'help
`mailx [opzioni] [user]`	
`Mail [opzioni] [user]`	
`man [opzioni] comando`	mostra la pagina di manuale (*man*) del comando specificato
`mkdir [opzioni] directory`	crea una directory
`more [opzioni] file`	impaginatore di un file testuale
`less [opzioni] file`	
`pg [opzioni] file`	
`mv [opzioni] file1 file2`	muove `file1` in `file2`
`od [opzioni] file`	dump su un file binario, in ottale, in ASCII, in esadecimale, in decimale o in

Comando/Sintassi	Cosa fa
	modalità carattere.
`passwd [opzioni]`	setta o modifica la propria password
`paste [opzioni] file`	incolla campi nelle linee del file specificato
`pr [opzioni] file`	filtra il file e lo stampa su un terminale
`ps [opzioni]`	mostra lo stato dei processi attivi
`pwd`	stampa la directory di lavoro (corrente)
`rcp [opzioni] nomehost`	copia file in remoto dalla macchina corrente in un'altra macchina
`rlogin [opzioni] nomehost`	effettua il login in remoto su un'altra macchina
`rm [opzioni] file`	rimuove (elimina) un file o una directory (`-r` elimina ricorsivamente la directory ed il suo contenuto) (`-i`

Comando/Sintassi	Cosa fa
	chiede conferma prima di eliminare i file)
`rmdir [opzioni] directory`	rimuove una directory
`rsh [opzioni] nomehost`	shell remota da eseguire su un'altra macchina
`script file`	memorizza ogni cosa che appare sullo schermo in un file fino a quando si esegue `exit`
`sed [opzioni] file`	editor di flusso per editare file da uno script o da linea di comando
`sort [opzioni] file`	ordina le linee del file specificato basandosi sulle opzioni scelte
`source file`	legge i comandi dal file specificato e li esegue nella shell corrente. `source` per la shell `C`, `.` per
`. file`	

Comando/Sintassi	Cosa fa
	la shell Bourne
`strings [opzioni] file`	riporta sequenze di 4 o più caratteri stampabili terminanti in <NL> o <NULL>. Generalmente viene usato per ricercare in file binari stringhe ASCII
`stty [opzioni]`	setta o visualizza le opzioni del terminale di controllo
`tail [opzioni] file`	mostra le ultime linee (o parte) di un file
`tar [opzioni] [file]`	archivio a nastro--riferirsi alle pagine man per dettagli su come creare, elencare ed estrarre file da un archivio. I file tar possono essere memorizzati su disco o su nastro.
`tee [opzioni] file`	copia standard output

Comando/Sintassi	Cosa fa
	in uno o più file
`telnet [host [porta]]`	comunica con un altro host usando il protocollo telnet
`touch [opzioni] [data/ora] file`	crea un file vuoto o aggiorna la data/ora di accesso di un file esistente
`tr [opzioni] stringa1 stringa2`	traduce i caratteri di *`stringa1`* provenienti da standard input in quelli di *`stringa2`* per standard output
`uncompress file.Z`	decomprime il file specificato e lo salva in un file
`uniq [opzioni] file`	rimuove le linee ripetute in un file
`uudecode [file]`	decodifica un file codificato con `uuencode`, riportandolo al file originale
`uuencode [file] nuovo_nome`	codifica un file binario in codice AS-

Comando/Sintassi	Cosa fa
	CII 7-bit; utile quando lo si trasmette via email, per essere poi decodificato come `nuovo_nome` alla destinazione
`vi [opzioni] file`	editor visuale, a pieno schermo
`wc [opzioni] [file]`	mostra il numero di parole (o di caratteri o di linee) del file specificato
`whereis [opzioni] comando`	riporta le locazioni del binario, del sorgente e della pagina man per il comando specificato
`which comando`	riporta il percorso del comando o l'alias di shell in uso
`who` o `w`	riporta chi è «loggato» e quali processi ha in esecuzione
`zcat file.Z`	concatena (lista) il file non com-

Comando/Sintassi	Cosa fa
	presso a video, lasciando il file compresso su disco

Capitolo 12. Una breve bibliografia Unix

Tabella 12.1. Una breve bibliografia Unix

Titolo	Autori	Editore
UNIX: concetti, strutture, utilizzo	James R. Groff, Paul N. Weinberg	Jackson
Guida completa Unix System V. Release 4	Coffin Steven	Mc Graw Hill
Introduzione all'architettura di sistema Unix	Prabhat K. Andleigh	Jackson
Usare Unix senza fatica	John R. Levine, Margaret Levine Young	Mc Graw Hill
La grande guida Unix	John J. Valley	Jackson
Shell Unix guida alla programmazione	Arthur, Burns	Mc Graw Hill
Introduzione alla struttura interna di Unix	Le Van Huu	Unicopli
Unix guida completa	Rosen, Host, Farber, Rosinski	Mc Graw Hill
Gestione del sistema operativo Unix	Bruno Catalano, Ezio Raddi	Città Studi
Guida a Unix seconda edizione	Hahn H.	Mc Graw Hill
I segreti di Unix Seconda edizione	Armstrong James	Apogeo
Guida a Unix con Linux	Jack Dent, Tony Gaddis	Apogeo
Introduzione a Unix	David L. Schwartz	Apogeo
Unix manuale per l'amministratore di sistema - terza edizione	Nemeth Evi, Snyder Garth, Seebass Scott, Hein Trent	Pearson Education Italia

Glossario

A

account — struttura di dati che contiene alcune informazioni (come password, nome reale, username, indirizzo, ecc.) relative a un utente

alias — meccanismo che permette di invocare un comando (o una combinazione di comandi) attraverso un nome pseudonimo precedentemente assegnato

B

background — riferito alla modalità di gestione dell'esecuzione di un processo in shell corrente: la shell non aspetta la fine di tale processo ma continua la sua normale esecuzione

Vedere: sezione 5.5

backquote — caratteri speciali utilizzati per inserire in una stringa l'output del comando specificato tra questi caratteri

Vedere: sezione 9.2

built-in — comandi incorporati nel codice della shell

Vedere: sezione 5.1

C

case sensitive — distinzione tra lettere maiuscole e lettere minuscole

D

directory-home — è la directory dell'utente, nella quale ci si ritrova (in generale) dopo la fase di login

Vedere: sezione 3.6

directory-root — è la directory padre di tutte le altre directory (indicata spesso con il segno /)

Vedere: sezione 2.2

E

editor — strumenti per manipolare testi (programmi, ecc.)

Vedere: Capitolo 10

EOF — sta per «end of file» e indica la fine del file

F

filename — nome del file

file system — architettura formata da un insieme di strutture dati che permetto di organizzare e gestire i dati su disco

Vedere: sezione 2.2

foreground — riferito alla modalità di gestione dell'esecuzione di un processo in shell corrente: la shell aspetta che il processo termini prima di riprendere la sua esecuzione

Vedere: sezione 5.5

G

groupid — numero che identifica in maniera univoca un gruppo di utenti su un sistema Unix

Vedere: sezione 3.1.4

groupname — nome che identifica un gruppo su un sistema Unix

Vedere: sezione 3.1.4

H

hard link — link fisico tra due file

Vedere: sezione 8.1.7

here document — speciale forma di quoting

Vedere: sezione 9.6

history — storia dei comandi digitati precedentemente

Vedere: sezione 5.6

I

inode — struttura di dati contenente informazioni sui file

Vedere: sezione 2.3

J

job — sinonimo di processo

K

kernel — è il cuore del sistema operativo, il posto in cui sono caricati tutti i driver per i device, gestisce i processi, le operazione di I/O, ecc.

L

login — riferito sia al nome utente di un account Unix (login username) sia alla fase (fase di login) di accesso a un sistema Unix (login+password) (il riferimento risulta chiaro dal contesto)

Vedere: sezione 3.1

M

man page — pagina del manuale di un sistema Unix

Vedere: sezione 3.5

multi-level file system — file system a più livelli organizzati in gerarchia

Vedere: sezione 2.2

multi-processor — o multi-tasking; capacità del sistema operativo che permette di eseguire più programmi (processi) senza doverne aspettare la terminazione sequenziale

Vedere: sezione 2.1

multi-user — capacità del sistema operativo che permette di gestire più utenti contemporaneamente

Vedere: sezione 2.1

O

OS — operating system, sistema operativo

P

password — codice segreto di un account Unix utilizzato per autenticare un utente nel sistema

Vedere: sezione 3.1 e sezione 3.1.2

path name — percorso dei nomi, che indica i «rami» del file system (directory separate da un /) che bisogna attraversare per raggiungere un determinato file o directory

Vedere: sezione 2.2

pattern — schema

pipe/piping — meccanismo che porta l'output di un comando nell'input di un altro

Vedere: sezione 6.2

Q

quoting — tecnica di interpretazione di una stringa da parte della shell

Vedere: sezione 9.3

S

script — programma scritto nel linguaggio della shell

Vedere: sezione 9.1

shell — interprete a linea di comando. È l'interfaccia tra l'utente e il sistema operativo (kernel)

Vedere: Capitolo 5

simbolic link — link simbolico (collegamento tramite path name) tra due file o directory

Vedere: sezione 8.1.7

system call — sono delle procedure (routine) standard, che i programmi utente utilizzano per accedere ai servizi che il kernel mette a disposizione

Vedere: sezione 2.1

standard error — dispositivo standard di errori (terminale di default)

Vedere: sezione 6.1

standard input — dispositivo standard di ingresso (tastiera di default)

Vedere: sezione 6.1

standard output — dispositivo standard di uscita (terminale di default)

Vedere: sezione 6.1

super-user — o root, utente speciale che amministra un sistema Unix

U

userid — numero che identifica univocamente l'utente su un sistema Unix

Vedere: sezione 3.1.4

username — o user, nome di un utente Unix (utilizzato nella fase di login)

Vedere: sezione 3.1.4

V

verbose — modalità di esecuzione di un comando che fornisce molte informazioni addizionali (utile per gli utenti novizi)

W

wild card	anche conosciuti come meta-caratteri, sono caratteri speciali utilizzati in alcune circostanze per il confronto/sostituzione di caratteri *Vedere:* sezione 6.4
word-wrap	meccanismo che invia alla prossima riga l'intera parola fuori margine