1 L'interfaccia a carattere di Unix

1.1 La "shell"

Console di terminale.

Quando si possono dare comandi ad un S.O. non si comunica direttamente con il suo kernel, ma con un programma, che del S.O. fa parte, e che si chiama "command interpreter" o "shell" (conchiglia, dato che "isola" il S.O. dall'utente). In Unix la shell è un vero e proprio interprete di un linguaggio, che ha la sua sintassi, le sue variabili ed anche istruzioni per il controllo flusso.
Con le istruzioni dei linguaggi di comando Unix si possono scrivere dei "file batch" molto sofisticati, che possono diventare vere e proprie applicazioni. Questi "file batch" vengono detti "script" (command script o "shell script").
Unix ammette shell diverse, ciascuna con linguaggi di comando diversi. Molte shell hanno comandi molto simili. Ogni utente può usare quella che più gli piace; ne può usare anche più di una. La più comune shell è bash (**B**ourne Again Shell), altre sono: sh, ash, bsh, ksh, csh, tcsh.

1.2 Agevolazioni nell'uso dell'interprete di comandi

1.2.1 Tab

La pressione del tasto Tab completa i nomi di file e directory scritti solo in parte.

1.2.2 Tab Tab

La doppia pressione del tasto Tab visualizza i file eseguibili che possono essere lanciati automaticamente[1] e che cominciano con le lettere scritte fino a quando abbiamo premuto.

2 Comandi CUI di Unix, confronto con comandi DOS

DOS	Linux	Note
ATTRIB (+-) attr file	chmod <mode> file	completamente differenti
BACKUP	tar -Mcvf device dir/	completamente differenti
CD dirname\	cd dirname/	quasi uguali
COPY file1 file2	cp file1 file2	quasi uguali
DEL file	rm file	niente undelete in Unix, almeno a linea di comando (esiste un cestino nei desktop manager)!
DELTREE dirname	rm -R dirname/	"
DIR	ls	non del tutto uguali
DIR file /S	find . -iname file	completamente differenti, ma questi comandi danno gli stessi risultati
EDIT file	vi file	vi è scomodo (meglio mc)
FORMAT	mkfs fdformat mount umount	diversi
HELP command	man command info command	simile con ipertesto
MD dirname	mkdir dirname/	quasi uguali
MORE < file	less file	molto meglio

1 sono "nella path" (vedi oltre)

MOVE file1 file2	mv file1 file2	molto meglio
PRINT file	lpr file	
PRN	/dev/lp0 /dev/lp1	
RD dirname	rmdir dirname/	quasi uguali
REN file1 file2	mv file1 file2	non funziona con più di un file
RESTORE	tar -Mxpvf device	diversi
TYPE file	less file	molto meglio
WIN	startx	
/	-	per introdurre le opzioni dei comandi
\	/	per separare i directory nelle path

Tabella adattata da "DOS/Windows to Linux HOWTO" By Guido Gonzato

Comandi e nomi dei file sono **CaSE SeNsItIvE!**

2.1 Indicazione dei file e delle directory (cartelle)

In Unix il carattere di separazione fra i nomi dei percorsi (path) delle cartelle è "/". Quando fu

./ è la directory "corrente" (quella in cui l'interprete dei comandi è attualmente impostato)

../ è la directory "madre" di quella corrente

/ è la directory radice del sistema (directory root)

2.2 Nomi nel file system

In Unix i nomi dei file sono lunghi, senza estensioni "obbligatorie", molto spesso non ci sono convenzioni sull'estensione dei file. Un qualsiasi file, con o senza estensione, può essere eseguibile. Non dipende dall'estensione ma dagli attributi che gli vengono dati.
Se i nomi di file hanno spazi, è meglio metterli sempre fra virgolette:

```
$ mkdir "Nome directory con spazi"
```

Per quanto si possano dare nomi di file che comprendono degli spazi, è consigliabile usare nomi interi per i comandi ed i programmi.

Si usa "slash" ("/"[2]) per separare i directory nelle path.

2.2.1 man

Il comando "man" è il manuale dei comandi Unix. Dà una spiegazione, piuttosto dettagliata, del comando Linux che si passa come parametro. man spiega il significato di tutte le opzioni del comando, anche se di solito ci sono pochi esempi.
Per cercare una stringa con man: premere "/" e scrivere la stringa da cercare, poi premere Enter (Invio).
Per uscire da man: tasto "Q".
Esempio:

```
$ man man
```

visualizza il manuale del comando man (di sé stesso!).

Oggi in alcune distribuzioni Linux (s. Fedora) le pagine man possono anche essere viste da un browser Web, indicando come "URL", nel browser Web di sistema, "man:/" (con UNA sola barra).

2.2.2 info

Visualizza informazioni che riguardano il comando. Il programma è ipertestuale, si possono "seguire" dei link in modo non lineare. Le informazioni "ufficiali" sui comandi Unix realizzati da FSF (GNU) sono rilasciate tutte in formato "info". Anche i file info si possono consultare con un Web Browser.

2 Segno di divisione NON barra invertita (backslash). La convenzione è il contrario di quella di Windows.

2.2.3 echo <stringa>

Scrive la <stringa> sul monitor (sul "file" di console)

2.3 Comandi per i directory ed i filesystem

2.3.1 home directory

Ogni utente ha una "home" directory dalla quale comincia ad esplorare l'albero dei directory quando fa il login. Tipicamente le home directory risiedono sotto /home.

2.3.2 mkfs

make filesystem è equivalente al FORMAT dell'MS-DOS
Esempi:

```
$ mkfs -t ext2 -c /dev/fd0
```

formatta un dischetto in formato Linux

```
$ mkfs -t dos -c /dev/fd0
```

formatta un dischetto in formato DOS. Per formattare a basso livello un floppy disc si può usare fdformat.

2.3.3 Change Directory

cd <directory>

Fa cambiare la directory corrente.

2.3.4 pwd

Print **W**orking **D**irectory. Fa vedere la path completa della directory corrente.
Esempio:

```
$ pwd /usr/shared/programmi/gruppo1/DirFratello
```

2.3.5 mkdir <Directory>

"make directory", analogo a md di MS-DOS

```
$ mkdir ../DirFratello
```

Crea una nuova cartella "DirFratello", a partire dalla cartella corrente.

cd e **pwd (print working directory)**

```
$ cd ../DirFratello
$ pwd
/usr/shared/programmi/gruppo1/DirFratello
```

il comando "pwd" mostra l'intero percorso della cartella corrente (se ci si "perde" navigando nelle cartelle).

2.3.6 df (disk free)

Mostra lo spazio rimasto libero in tutte le partizioni attualmente montate.
Esempio:

```
$ df
Filesystem          Dimens.    Usati    Disp.    Uso% Montato su
/dev/hda5           4,9G       1,9G     2,8G     41%      /
/dev/hda1           17G        6,3G              72%      /C
/dev/hda7           5,9G       3,6G     2,3G     62%      /D
/dev/hda3           3,6G       2,1G     1,6G     58%      /w2003
```

2.3.7 du (disk used)

Mostra lo spazio occupato sul disco dal directory indicato e da tutti i suoi "figli". Le sue opzioni principali sono:

- **-a** (all) visualizza ogni file, non solo i directory
- **-h** (human) visualizza in modo "umano", con i prefissi k, M, G (attualmente è il default)
- **-s** (summarize) visualizza solo il totale per ogni argomento, evitando di visualizzare ogni sottodirectory
- **-x** (exclude) esclude le directory che sono su un filesystem diverso da quello dell'argomento
- **-c** (anche --total) alla fine di tutta la visualizzazione, mostra anche il totale di ciò che è stato visualizzato

Esempio:

```
$ du /usr -s
1,7G /usr
```

2.3.8 free

free visualizza la memoria libera e lo swap space libero.

2.3.9 rmdir <Directory>

"remove directory", analogo a rd di MS-DOS

```
$ rm * -r
```

cancella in modo ricorsivo (-r) tutti i file di questa cartella e di tutte quelle che vi sono contenute!

2.3.10 echo

echo <stringa>

Scrive la <stringa> nel "file" di console. E' molto utile negli script, per visualizzare qualcosa sul monitor.

2.3.11 ls [<Nome File>] (list files)

comando "list files", è analogo a dir, ha molte opzioni, le più importanti:

- **-l** (long format) dettaglio delle informazioni sui file, dà anche proprietari e diritti sui file
- **-t** (time) in ordine di creazione (prima i più recenti)
- **-u** (used) prima quelli usati più di recente
- **-r** (reverse) in ordine inverso
- **-h** (human) mostra le dimensioni dei file in modo "umano" (in kByte e non in"blocchi" di 512 Byte)
- **-d** (directory) mostra solo quei file che sono directory
- **--color** output a colori, nel quale si distingue facilmente il "tipo" dei file.

Si può usare anche il comando "dir", che non è altro che il comando ls chiamato con opzioni di default diverse.
dir [<Nome File>] è analogo al comando dir di MS-DOS. Al posto di un singolo nome ci può essere uno schema che indica molti file (filename expansion).
ls e dir non fanno vedere il nome dei file che iniziano con il punto (filenascosti). Per vedere quei file bisogna usare l'opzione -a.

2.3.12 Wildcard

Negli "schemi di selezione" dei comandi Unix la shell permette di abbreviare la scrittura del nome dei file in ogni punto dove "serve" (schema per la selezione multipla dei file). L'abbreviazione avviene utilizzando i "caratteri jolly" (**wildcard**). Il carattere "*" sostituisce tutti i caratteri nel nome di un file, mentre "?" ne sostituisce solo uno.
Esempi: se abbiamo in un directory:

```
$ ls
text1.txt text2.txt text3.txt esercizio prova.txt text13.txt esempio compresso.-
txt.gz elicottero
```

allora:

```
$ ls *.txt
text1.txt text2.txt text3.txt prova.txt text13.txt
$ ls *.gz
compresso.txt.gz
$ ls *.txt.gz
compresso.txt.gz
$ ls ese*
esercizio esempio
$ ls e*o
elicottero esercizio
$ ls text?.txt text1.txt text2.txt text3.txt
```

nota: non compare text13.txt ***** asterisco = vanno bene tutti i caratteri nel nome di file in numero qualunque
Altri esempi:

```
'ini*' seleziona tutti i nomi di file che iniziano con la stringa "ini"
'*ini*' seleziona tutti i nomi che contengono la stringa "ini"
```

? punto interrogativo = qualsiasi SINGOLO carattere
Esempi:

```
vendite??.dat seleziona vendite00.dat, vendite99.dat, ma non vendite2000.dat
```

[ ] parentesi quadre = contengono un elenco di caratteri che "vanno bene":
Esempi:
$ `vendite[90][90].dat`
seleziona
`vendite90.dat, vendite09.dat`
ma non
`vendite98.dat, vendite999.dat`

FARE ATTENZIONE A MODIFICARE I FILE CON LE WILDCARD, si potrebbero fare danni, specialmente quando si è "root"!

Attributi dei file: formato di ls -l

Il comando list con l'opzione -l (long) mostra tutti i dettagli delle informazioni sui file elencati. Le prime di queste informazioni sono gli attributi dei file, che indicano il "tipo" di file, ed i diritti degli utenti sul file.
Primo carattere: - = file normale, d = directory, l = link (collegamento)
I successivi caratteri si suddividono in gruppi di tre:
diritti dell'utente (r = read, w = write, x = execute, nell'ordine)
diritti del gruppo cui appartiene l'utente (come nei diritti dell'utente)
diritti di tutti gli altri (come nei diritti dell'utente)

2.3.13 "catenate"

cat <NomeFile>
senza opzioni funziona come TYPE di MS-DOS. Infatti visualizza il contenuto del file.
Se si passano a "cat" due o più file, il comando li concatena, cioè li scrive di seguito,prima il primo, poi il secondo.
Esempio:
`$ cat inferno.txt purgatorio.txt paradiso.txt`
visualizza sullo schermo tutto il testo di inferno.txt, poi tutto il testo di purgatorio.txt ed infine paradiso.txt.

Questo comando è utile per concatenare più file in uno complessivo, in questo modo:
`$ cat inferno.txt purgatorio.txt paradiso.txt > Commedia.txt`
produce il file "Commedia.txt" in cui le tre cantiche di Dante sono scritte in successione.

tree

Mostra l'albero dei directory

free

visualizza la memoria libera e lo swap space libero, invece **$ du** ("disk used") lo spazio usato nei dischi

2.3.14 copy

cp <Sorgente><Destinazione>

E' il comando di "copy", che non "cambia nome" durante la copia! La destinazione può essere un file od un directory. Se ci sono più di un file di sorgente, la destinazione DEVE essere un directory.
Opzioni importanti:
-d (no dereference) Durante la copia mantiene uguali i collegamenti simbolici, 'che altrimenti sarebbero sostituiti dai file cui si riferiscono.
-p (preserve) Mantiene le proprietà e i permessi dei file che copia
-R (Recurse subdirectories) copia anche le sottodirectory in modo "ricorsivo"
-u (update) copia solo i file che sono più recenti di quelli dello stesso nome già esistenti nel directory

Esempi:
`$ cp -dpR ./vecchiodir ./nuovodir`
copia in modo "identico" il directory vecchiodir in nuovodir. In nuovodir compare il directory vecchiodir.
`$ cp -dpR ./vecchiodir/* ./nuovodir`
copia, nello stesso modo, tutti i file di vecchiodir in nuovodir. In nuovodir non compare il directory vecchiodir, ma direttamente i suoi "figli".

2.3.15 remove

rm <schema nome files>
Cancella il file od il gruppo di file specificati dallo schema.
-r (recurse) con questa opzione si cancella anche dentro i directory. Non chiede conferma e lavora automaticamente. Attenzione ad usarlo si potrebbero fare danni enormi, soprattutto quando si è root!

2.3.16 file

file <schema nome files>

fa vedere il tipo dei file il cui nome rispetta lo schema. Si può usare per "provare" la rm priva di eseguirla, per evitare danni.

2.3.17 move

Sposta i file indicati in un altro directory. Se usata nello stesso directory cambia il nome del file.

mv <schema che indica i file sorgente> <directory di destinazione>
oppure
mv <nome di file sorgente> <nome di file destinazione>
se il directory sorgente e destinazione sono lo stesso con mv si cambia anche nome ai file:
mv <vecchio nome> <nuovo nome>
Visto come funziona mv, si capisce che non si può cambiare nome a gruppi di file con le "wildcard", per questo si deve usare il comando rename.

2.3.18 Rinomina gruppi di file

rename <lettere da sostituire> <lettere con cui sostituire> <schema che indica i file>
Esempi:

```
$ rename va va.old prova
```

cambia il nome del file prova in prova.old

```
$ rename htm html *.htm
```

sostituisce htm con html in tutti i file il cui nome termina con .htm. La sostituzione avviene alla prima occorrenza, per cui il file FILEhtm.htm viene rinominato come: FILEhtml.htm, che non è quello che si voleva. Se si vuole essere sicuri della sostituzione delle estensioni anche in questo file si può fare così:

```
$ rename .htm .html *.htm
```

2.4 Collegamenti

ln <file da collegare> <nome del link>

Un collegamento è un riferimento (link) ad un file, il software si comporta come se fosse un "vero" file, ma in realtà è solo un "rimando" alla posizione vera del file.
I link puntano ad altri file e possono essere usati al loro posto. Con un collegamento il nome del file viene visto nei directory e dal command processor come se fosse il file originario.

Esempio:

```
# ln -s linux-2.4.0 linux
```

collega il file linux-2.4.0 con il nome simbolico linux (entrambi nella stessa directory). Dopo questo comando usare il file linux equivale ad usare il file linux-2.4.0
-s fa fare un link simbolico e non "hard".

Esempio: i due comandi useradd adduser sono di fatto la stessa cosa: uno è un link simbolico all'altro.

2.5 Variabili d'ambiente

L'interprete di comandi può creare alcune variabili, entro le quali memorizza stringhe che gli serviranno in seguito. Dettagli sulle variabili d'ambiente verrano dati in seguito.

2.6 Percorsi per la ricerca degli eseguibili (variabile path)

I programmi che possono partire senza indicare tutto il percorso (path) del file eseguibile sono all'interno di una delle directory comprese nell'elenco specificato dalla variabile d'ambiente path,

echo $path

A differenza di DOS, la directory corrente (./) NON è nella path degli eseguibili, perciò per far partire un programma può essere necessario indicare che esso deve partire dalla directory corrente; se il file eseguibile "programma" è nella directory locale e non è in $path la prima di queste NON lo fa partire, l'altra sì:

programma

./programma

2.6.1 Cambio degli attributi dei file

chmod [<Opzioni> <Modo>[,<Modo>..]] <file> sdfdsf

"change mode": cambia i permessi d'accesso dei file. I modi preceduti da + aggiungono un permesso, quelli preceduti da - lo tolgono.

Esempio:

```
$ chmod 'ugo+x' ScriptCancellazione
```

Trasforma il file ScriptCancellazione in eseguibile, sia per l'utente che lo possiede (u) che per il il suo gruppo (g) e per tutti gli altri (o = other). Dopo questo comando ogni utente è in grado di eseguire quel file, per esempio così:

```
# ./ScriptCancellazione
```

per dare attributi diversi ai vari soggetti si può' usare la virgola:

```
$ chmod 'u+x', 'g-r', 'o=x', ScriptCancellazione
```

in questo modo si aggiunge il diritto di esecuzione all'utente, si toglie il diritto di lettura al gruppo e si lascia il SOLO il diritto di esecuzione agli altri.
il diritto x per i file e' il diritto di esecuzione, per le directory e' il diritto di accesso.

chmod in modo "ottale"

esiste una forma di chmod più sofisticata, ma più facile da usare. Considerando l'ordine con cui sono visualizzati i diritti di accesso nel comando ls -l, si può sostituire un numero ottale (con cifre da 0 a 7) a ciascuno dei diritti di: owner, group ed altri. Il numero viene formato mettendo un 1 per ogni diritto che si vuole accordare.
Esempi di numeri:

```
0 = 000 = ---    corrisponde a "nessun diritto"
7 = 111 = rwx    corrisponde a tutti i diritti: lett.(r), scritt. (w), esecuzione (x)
5 = 101 = r-x    corrisponde a diritti di lettura ed esecuzione
```

Esempi di comandi:

```
$ chmod 750 programma
```

Dà tutti i diritti (7) all'"owner" del file "programma", diritti di lettura ed esecuzione al gruppo del file e nessun diritto (0) agli altri.

2.6.2 which

which <file>
Fa vedere la path di un eseguibile che parte senza specificarla, perchè è in una directory inclusa nella variabile $path della shell (per le variabili della shell, vedi oltre)

2.6.3 whereis

whereis <file>
Fa vedere la path dell'**eseguibile**, quella del **sorgente** e quella che contiene la pagina di **manuale** del programma indicato.

2.6.4 Ricerca di un file per nome

find <Path> <Espressione>

<Path> è il directory dal quale si parte per la ricerca. <Espressione> è lo schema che permette di trovare il file cercato, esprimendo le opzioni volute. find ha molte opzioni, con le quali si può limitare la ricerca nei "dintorni" di un certo directory, si possono escludere i file system "stranieri", trovare i file creati, letti o modificati negli ultimi n minuti, ore o giorni, trovare solo i file più grandi o più piccoli di una certa soglia, i file appartenenti ad un certo utente o quelli che hanno certi attributi. Nelle espressioni che determinano il risultato della ricerca è possibile usare gli operatori logici AND e OR.

Opzioni più importanti:

- -name
  Per la ricerca del nome si può usare l'opzione -name (o -iname), consente la ricerca con i caratteri jolly (wildcard), del tutto analoga alla ricerca dei file in MSDOS. -name è case sensitive, -iname è case insensitive
- -regex
  ricerca con una "regular expression", analoga a quella del comando grep (vedi).
- -prune
  ignora tutti i directory che discendono da quello specificato come path di ricerca

- -maxdepth <numero> <lettera>
  <numero> indica di quanti livelli si deve scendere in profondità nella ricorsione dei directory, <lettera> seleziona i file da mostrare: se <lettera>= d cerca fra i directory se <lettera>= f cerca fra i file "normali" se <lettera>= l cerca fra i link simbolici
- -xdev
  non discende nei sottodirectory "stranieri", che hanno filesytem diversi da quello da cui si comincia (p.es. dos)
- -fstype <tipo di filesystem>
  discende solo nei directory che hanno il tipo di filesystem indicato.

Esempi:
```
$ find . -name '*bash*' -type d -xdev -ls
```
. "punto" è il directory corrente (se non si mette . find non funziona!); il comando cerca dal directory corrente (.) i directory (-type d) il cui nome "case sensitive" (-name) comprende "bash", escludendo dalla ricerca (-xdev) i filesystem non Linux (es. FAT32 o NTFS). L'opzione -ls indica che il formato delle righe trovate è quello del comando ls (list directory) .
```
$ find / -fstype vfat -iname '*.bat'
```
cerca nei filesystem di tipo Windows FAT32 tutti i file che hanno estensione .BAT, la ricerca è "case insensitive".
```
$ find . -name 'sec*'
./securetty ./security
$ find . -name '*sec*'
./securetty ./ppp/chap-secrets ./ppp/pap-secrets ./security
```

2.6.5 Ricerca nel contenuto dei file

grep <regular expression> <file>

"get regular espression" serve per cercare all'interno del contenuto dei file; visualizza tutte le righe di <file> che contengono uno schema da ricercare (pattern), detto "regular expression" (espressione regolare). <file> può essere uno schema con wildcard, che indica file multipli. grep cercherà in tutti i file indicati dallo schema. La <regular expression> è lo schema di ricerca usato da grep. Esso NON funziona come quello usato per i file, ed è molto più potente.
Esempi semplici:
```
$ grep MONTI elenco_insegnanti.txt
```
fa vedere tutte le linee di elenco_insegnanti.txt in cui c'è scritta la parola MONTI (con lettere tutte maiuscole).
```
$ grep MONTI *
```
mostra le linee in cui c'è scritto MONTI in tutti i file del directory corrente. Non è necessario specificare ./ Infatti grep lavora per default nel directory corrente e la precedente è uguale a:
```
$ grep MONTI ./*
```
In un'espressione regolare il punto indica un qualsiasi carattere.
$ grep . ./prova
Visualizza tutte le linee del file prova, tranne quelle che non comprendono neppure un carattere. Infatti l'espressione regolare viene verificata per ogni riga che abbia almeno un carattere qualsiasi.
Alcuni caratteri dell'espressione regolare possono essere intercettati dall'interprete di comandi, se si vogliono usare quei caratteri si può mettere la stringa che si ricerca fra virgolette. Per esempio, se voglio cercare in tutti i file le linee in cui c'è il carattere "*", posso pensare di far così:
```
$ grep & *
```
però la cosa non funziona, perchè l'interprete "interpreta" & (che è il carattere per il lancio di applicazioni in background) e tenta di lanciare grep in background. Per cercare le linee con & si deve fare così:
```
$ grep '&' *
```
Per stare sul sicuro dunque, è meglio abituarsi a mettere le virgolette (meglio semplici, doppie solo se si fa quello ce si fa).
Le più importanti opzioni di grep:

Opzione	Descrizione
-F	Utilizza un modello fatto di stringhe fisse (lo schema di ricerca è una semplice stringa e non un'espressione regolare)
-C<numero>	stampa <numero> linee prima e dopo (context); se <numero>è omesso ne stampa due (fra C e <numero> non bisogna lasciare spazi)
-A<numero>	oltre alla linea in cui viene trovato ciò che si cerca, stampa anche il numero di linee successive indicato da <numero> (After context)

Esistono molte altre opzioni (consultare la man page).

<regular expression>
Un'espressione regolare definisce un criterio tramite il quale le stringhe da ricercare vengono accettate. E' composta di una sequenza di caratteri che debbono corrispondere nella stringa da cercare. Della sequenza possono far parte altre sequenze racchiuse fra parentesi quadre.
In una espressione regolare ogni sequenza di caratteri compresa fra parentesi quadre corrisponde ad un carattere nella stringa da cercare. Se all'inizio della sequenza fra parentesi quadre c'è il carattere "^", il significato della espressione viene invertito:
Esempio:
`[1245]` seleziona un singolo carattere qualsiasi fra "1" e "5", "3" escluso
`[^1245]` seleziona un singolo qualsiasi carattere, esclusi quelli che vanno da "1" a "5", il "3" invece è compreso.
`[.]` il punto fra parentesi quadre indica un solo carattere qualsiasi

```
$ grep A[1245] sorgente.txt
```

mostra le linee del file sorgente.txt che contengono A1, A2, A4, A5.
Invece che tutti i caratteri si può indicare un intervallo, con il simbolo "-"
Esempio:
`[1-5]` è uguale a [12345]
`[A-Z]` è una qualsiasi lettera maiuscola
`[A-z]` è una qualsiasi lettera, ma anche uno qualsiasi dei caratteri che hanno codici ASCII che stanno fra "A" e "z".
Esistono certi tipi di caratteri cui è stato dato un nome, che deve essere racchiuso fra due punti:
`[:alpha:]` carattere alfabetico, [:alpha:] = [A-Za-Z]
`[:alnum:]` carattere alfanumerico, [:alnum:] = [0-9A-Za-Z]
`[:cntrl:]` carattere di controllo (ASCII < 32)
`[:digit:]` cifra
`[:upper:]` lettera maiuscola
`[:lower:]` lettera minuscola
`[:punct:]` segno di interpunzione
`[:space:]` spazio (blank)
Quando è dentro le parentesi quadre il simbolo ^ inverte il senso di ciò che è specificato nelle parentesi; se invece il simbolo ^ è al di fuori delle parentesi quadre esso significa "stringa vuota all'inizio della linea". Per cui l'espressione regolare: ^Nel trova tutti i pattern che iniziano per "Nel".
Analogo è il simbolo $, che, se scritto in fondo all'espressione regolare, significa "la stringa nulla in fondo alla linea".
Perciò se vogliamo le righe dell'inferno che terminano per la parola stelle:

```
$ grep stelle$ inferno.txt
e 'l sol montava 'n su con quelle stelle
per sua dimora; onde a guardar le stelle
```

Ripetizioni in espressioni regolari

Codifica	Corrispondenza
x{n}	il carattere x è ripetuto esattamente n volte
x{n,}	x è ripetuto almeno n volte
x{n,m}	x è ripetuto da n a m volte
x*	x è ripetuto nessuna o più volte. Equivalente a x{0,}
x+	x ripetuto una o più volte. Equivalente a x{1,}
x?	x è ripetuto nessuna o al massimo una volta. Equivalente a x{0,1}

Per cui: .*alfa trova tutte le linee che contengono la parola alfa (.* significa "qualsiasi carattere (.) ripetuto nessuna o più volte (*)"). Vengono trovate: alfanumerico alfa romeo raggi alfa .+alfa delle precedenti stringhe trova solo: raggi alfa mentre non trova: alfanumerico alfa romeo perchè il + chiede che ci sia almeno un carattere prima di "alfa".
Se si devono cercare caratteri come :, +, ? e tutti gli altri che nelle espressioni regolari hanno significati particolari, bisogna farli precedere da \ (backslash, carattere di escape):
Esempio:
Se vogliamo solo le occorrenze di "stelle" che sono seguite da un punto, in inferno.txt, possiamo fare così:

```
# grep stelle[\.] inferno.txt
```

E quindi uscimmo a riveder le stelle.

```
$ grep a[\[] ./*
```

cerca la sottostringa "a["

```
$ grep [\ \!]eof *
```

cerca le occorrenze di " eof" (blank seguito da eof) o di "!eof" in tutti i file della directory corrente.

```
$ grep -E -e "alt[aoei]" dati.txt
```

cerca l'occorrenza di: alta, alto, alte, alti, nel file dati.txt

```
$ grep caron inferno.txt
```

non dà nulla mentre:

```
$ grep -i caron inferno.txt
```

dà tutte le righe in cui si nomina Caronte, che viene giustamente scritto con la prima lettera maiuscola.

```
$ ls -l | grep ^d[r-]
```

fa vedere tutti i directory (ed eventuali altre righe che comincino per dr o d-); il comando funziona così: **ls -l** fa vedere un "listato" completo dei file, con indicati i suoi flag. Se il primo dei flag indicati vale d, il file è una directory esso è seguito dal permesso di scrittura, che è indicato da w, oppure da -. Dopo il segno di "pipe" (vedi oltre) segue grep che visualizza tutte le linee del directory che comprendono dr oppure d-.

Esempio: Supponiamo di voler trovare tutte le occorrenze della parola "stelle" nel file inferno.txt:

```
$ grep -n stelle inferno.txt
46:e 'l sol montava 'n sù con quelle stelle
319:risonavan per l'aere sanza stelle,
2188:e torni a riveder le belle stelle,
2716:per sua dimora; onde a guardar le stelle
3693: Tutte le stelle già de l'altro polo
4894: E quindi uscimmo a riveder le stelle.
```

Ove il suo numero precede ogni linea trovata. Dunque vengono individuate 6 linee. Se vogliamo contare anche le "stella":

```
$ grep -in stell[ae] inferno.txt | wc -l
13
```

Il risultato di grep viene mandato a wc (word count) che con l'opzione -l conteggia il numero di linee. Ci sono perciò 11 linee che contengono la parola "stella" oppure "stelle". Le 7 nuove linee dovrebbero contenere "stella". Però, se guardiamo bene, troviamo le seguenti due linee (omettiamo l'elenco completo): 2953: con tamburi e con cenni di castella, 4679:d'aver tradita te de le castella, dove si trova "castella" e non "stella". Queste linee non dovrebbero essere conteggiate. Per eliminarle cambiamo lo schema, riconoscendo solo le righe in cui compare "stell[ae]" senza lettere prima o dopo.

Già che ci siamo eliminiamo anche le parole come "stellato":

```
$ grep [^a-zA-z]stell[ae][^:alpha:] inferno.txt | wc -l
11
```

[^a-zA-z] significa che il carattere NON deve essere fra a e z, maiuscole e minuscole, per cui prima e dopo stell[ae] ci deve essere una "non lettera".

[^a-zA-z] è la stessa cosa di [^:alpha:]. Ci rendiamo conto che potrebbero esserci scappate alcune "Stelle", con la prima lettera maiuscola, per cui, se aggiungiamo l'opzione -i:

```
$ grep -n -i [^a-zA-z]stell[ae][^a-zA-z] inferno.txt | wc -l
11
```

Concludiamo che nell'Inferno la parola stelle non è mai maiuscola. Si può notare che nel precedente comando invece di -n -i si potrebbe scrivere -ni "raggruppando" le opzioni sotto lo stesso "segno meno" (questo è vero anche per gli altri comandi Unix!). Per il problema specifico la ricerca avrebbe potuto essere molto più semplice, dato che si poteva cercare le linee in cui l'espressione regolare è una parola intera (opzione -w di grep, whole word):

```
$ grep stell[ae] inferno.txt -w | wc -l
11
```

Da notare che la seguente grep non funziona come ci si può aspettare ad una prima disamina:

```
$ grep -n stelle. inferno.txt
319:risonavan per l'aere sanza stelle
2188:e torni a riveder le belle stelle
3693: Tutte le stelle già de l'altro polo
4894: E quindi uscimmo a riveder le stelle.
```

Dato che . significa "qualsiasi carattere" vengono visualizzate le linee in cui "stelle" è seguito da "qualcosa". Infatti non sono presenti le linee 46 e 2716! Se vogliamo solo le linee in cui "stelle" compare alla fine:

```
$ grep -n stelle$ inferno.txt
46:e 'l sol montava 'n sù con quelle stelle
2716:per sua dimora; onde a guardar le stelle
```

Se vogliamo che sia l'ultima parola della linea, ma possa essere seguita da un numero qualsiasi di caratteri non alfabetici:

```
$ grep -n stelle[^:alfa:]*$ inferno.txt
46:e 'l sol montava 'n sù con quelle stelle
319:risonavan per l'aere sanza stelle,
2188:e torni a riveder le belle stelle,
2716:per sua dimora; onde a guardar le stelle
4894: E quindi uscimmo a riveder le stelle.
```

questa lista comprende tutte le occorrenze della parola stelle, seguite da un "non alfanumerico" [^:alfa:], ripetuto zero o più volte (*); poi deve seguire il fine linea ($); se arrivano altre lettere la linea non è valida. Per questo vengono accettate le righe che terminano con "stelle", "stelle,", "stelle.", ma viene esclusa la linea 3693, che contiene "stelle già".

Un altro esempio di uso di grep è il seguente:
```
# ls -lR / | grep ^[^d]*" "core$
```
Che trova tutti i file che si chiamano "core" e non sono directory, in tutti i filesystem montati. Infatti grep seleziona quelle linee dei directory che iniziano con un carattere diverso da d e terminano con ""core. La cosa funziona dato che ls -l scrive il nome del file come ultima cosa della linea. Naturalmente al posto della precedente si poteva usare:
```
# find / -name "core" -type f -fstype ext2
```
che, oltre ad esser più semplice da capire, dà un risultato più informativo e rapido.

Ancora un esempio:
Vogliamo cercare tutti i "tag" "FONT" in tutti i file html nel directory corrente . La prima cosa che viene in mente è la seguente:
```
# grep <FONT./*> ./*.html
```
questa grep cerca in TUTTA la linea in cui si trova un tag FONT; se però accade che ci sia una linea come la seguente:
```
piccolo <FONT size=4><FONT Face="Arial">GRANDE
```
vengono selezionati ENTRAMBI i tag della linea, non solo il primo. Se la grep serve per sostituire il tag con un altro essa non potrà funzionare. Questa grep viene interpreta come "cerca tutti i caratteri, ">" compreso, fino a che non si trova l'ULTIMO ">" della riga. Per selezionare solo il primo tag delle linea bisogna fare così:
```
# grep <FONT[^>]*> ./*.html
```
La riga viene visualizzata come prima, ma questa volta viene individuato il PRIMO dei due tag con FONT. Infatti la grep si può spiegare così: cerca la stringa "<FONT", poi prosegui se il carattere non è ">". Per questo alla PRIMA occorrenza di ">" la ricerca della regular expression si interrompe e viene individuato un solo tag nella linea.
Vediamo ora come individuare i tag font "di chiusura", che hanno lo slash (</FONT>). Il problema è che "/" è un carattere "di comando" delle regular expression, per cui esso deve essere preceduto da un "back slash" di escape:
```
# grep <\/FONT[^>]*> ./*.html
```
se invece voglio trovare sia i tag <FONT> che </FONT>, devo fare così:
```
# grep <\/?code[^>]*>
```
La presenza del punto interrogativo significa "zero o una occorrenza del carattere che precede", cioè dello slash "/", che deve essere preceduto dal back slash di "escape".

Le regular expression vengono usate anche in programmi di editing, che permettono di modificare il contenuto dei file, come "sed" e "awk", ma anche in molti text editor in ambiente a finestre, che hanno la ricerca per regularexpression (es. TexpPad). In questi casi è utile disporre di un modo per identificare , tutti insieme, i caratteri che vengono riconosciuti con una regular expression.
In genere per questo scopo viene usato il carattere "&", che, quando si fa una sostituzione significa "tutti i caratteri riconosciuti attraverso la regular expression di ricerca".
Se, per esempio, vogliamo cercare tutte le stringhe che siano fatte così:
!<qualsiasi carattere> =
e sostituirle con
!<gli stessi caratteri di prima> = Trim(
Dobbiamo cercare con la seguente regular expression

```
![^=]*=
```

e sostituire con questa:

```
& Trim(
```

L'editor che useremo aggiungerà a destra di tutte le ricorrenze della regular expression la stringa Trim(, per esempio:

```
Articolo!Codice = txtCodice
Cliente!RagioneSociale = txtNome
```

verranno sostituiti da:

```
Articolo!Codice = Trim(txtCodice
Cliente!RagioneSociale = Trim(txtNome
```

2.7 more

Fa fermare l'output quando si arriva in fondo ad una schermata. Quando ciò accade appare la scritta "more" e si attende fino alla pressione del tasto "space".

2.8 less

Simile a more, ma si può anche tornare indietro e cercare una stringa nel file. less comincia a visualizzare il risultato prima di finire di leggere il file, per cui è adatto a visualizzare i risultati di lunghe elaborazioni. Ha diversi comandi, che permettono di spostare la "pagina" visualizzata. Ha centinaia di opzioni, i comandi sono simili a quelli dell'editor vi. Scrivendo 10% va al punto che corrisponde al 10% del file. /(senza virgolette) cerca all'avanti ?cerca all'indietro n e N ripetono l'ultima ricerca di stringa per uscire premere q o Q + Enter. Il comando man usa less per visualizzare il manuale.

2.9 history

Visualizza la "storia" passata dei comandi di shell.
`$ history 10`
fa vedere gli ultimi 10 comandi. Se si cerca un particolare comando si può fare: `$ history | less` per potersi spostare liberamente

2.9.1 Comando script

`$ script`
apre una sessione di shell e comincia a scrivere nel file "typescript" tutto ciò che passa per il terminale. Per concludere il "log" di tutti i dati chiudere la shell con exit.

2.10 Manipolazione dei file di ingresso e di uscita dei programmi

I programmi CUI Unix hanno prendono i loro dati da un "file virtuale" (flusso di dati o stream) e mettono i risultati in un secondo "file virtuale". Il flusso in ingresso al programma viene detto "input stream" quello d'uscita "output stream". Normalmente l'input stream è la tastiera ("standard input") mentre l'output stream è il monitor a caratteri (console). Esiste inoltre un secondo flusso di output, detto "error stream", nel quale finiscono tutte le indicazioni di errore prodotte dal programma. Normalmente l'error stream è gettato, via, ma lo si può ridirigere sulla console od in un file "vero" per scopi di diagnostica.

2.10.1 >

Ridirezione dell'output (normalmente è la console), esempio:

```
$ echo 'ls -cChpl --color=auto' > l
```

crea il file "l", nel quale scrive la stringa 'ls -cChpl --color=auto'.

2.10.2 >>

Ridirezione dell'output con accodamento in fondo al file

2.10.3 <

Ridirezione dell'input. Lo standard input è normalmente è la tastiera; un programma "scritto per la tastiera" con il comando di ridirezione "<" prende il suo ingresso da un file.

Esempio:

```
# wc < paradiso.text
4929 31624 182418
```

Fornisce come risultato diversi conteggi sul contenuto del file "paradiso.text". Il comando wc utilizza come suo ingresso il file "paradiso.text"; è come se si scrivesse in tastiera tutto ciò che è contenuto in "paradiso.text" e lo si passasse a "wc". Per dettagli sul comando wc (word count), vedi oltre.

2.10.4 |

Pipe. La barra verticale indica la concatenazione di comandi. Il primo comando scritto viene eseguito, il suo risultato viene memorizzato temporaneamente in un file dal S.O., poi quel file temporaneo viene usato come ingresso del secondo comando.

Esempi:

```
$ cat FileLungo | less
```

Visualizza "a videate" il file "FileLungo", utilizzando il comando less. Il funzionamento del comando è il seguente:

1. Viene eseguito il comando "cat FileLungo"
2. Il risultato del comando "cat FileLungo", che andrebbe normalmente sullo standard output, viene invece messo in un file temporaneo

3. Il file temporaneo viene ridiretto come input del comando less
4. Il comando less visualizza il file "a videate"

```
# netstat -a | wc -l
```
conta le righe prodotte dal comando netstat -a, cioè il numero di servizi internet aperti sul computer[3].

Lancio di applicazioni in background

Se dopo un comando o il nome di un'applicazione si inserisce la lettera & l'interprete torna al prompt prima di terminare l'esecuzione del comando indicato. Se per esempio vogliamo lanciare un programma lungo e cominciare subito a fare altre cose possiamo fare così:

```
$ cat GuerraEpace.txt | sort > ordinato.txt &
[1] 924
$
```

3 Script di comandi (shell script)

3.1 Parametri degli script

```
$ echo 'ls -cChpl --color=auto $1' > l
```
come prima, abbiamo semplicemente aggiunto $1 come parametro "simbolico" di ls.

Ora la prima "parola" che viene passata nella linea di comando dopo l viene utilizzata "al posto" di $1.
il comando:

```
$ ./l m*.txt
```

equivale a scrivere:
```
$ ls -cChpl --color=auto m*.txt
```

se avessimo voluto accettare un **qualsiasi numero di parametri** per ls avremmo potuto usare la wildcard "*****" nello script:
```
$ echo 'ls -cChpl --color=auto $*' > l
```
siccome c'è $* al posto di $1, lo script accetta tutti i parametri che sono passati e li usa nel punto indicato.
per esempio:
```
$ ./l m*.txt -S
```

visualizza in ordine di dimensione dei file, e non in ordine di creazione, come indicato dal parametro -c.

3.2 Variabili d'ambiente

L'interprete di comandi può avere vere e proprie variabili, che sono stringhe definibili dall'utente.
Per assegnare il valore di una variabile basta usare l'uguale:
```
$ PATH=/sbin:/bin/:usr/sbin/:/usr/bin
```
nell'assegnazione non ci devono essere spazi, né prima, né dopo l'uguale.
Esempio:
```
$ echo $PATH
/sbin:/bin/:usr/sbin/:/usr/bin
```
Per vedere il valore corrente di tutte le variabili d'ambiente, digitare il comando set:
```
$ set
```

3.3 Filtri

Programmi che vengono usati tipicamente insieme ad altri, con uno dei meccanismi di ridIrezione degli ingressi e delle uscite che abbiamo già visto.

sort: ordina o fonde ordinando (merge) i file che costituiscono il suo ingresso.

Esempi:

```
# cat lista
Battista
Alberti
Leon

# sort lista
Alberti
```

3 Approssimativamente, non dà proprio il numero esatto, per via del formato del comando netstat

```
Battista
Leon

# cat lista1
Zorba
Catena
Abati

# sort lista lista1
Abati
Alberti
Battista
Catena
Leon
Zorba
```

tail-<numero righe> fa vedere le ultime <numero righe> linee del (o dei) file specificato

wc <file> "**w**ord **c**ount" mostra il numero di linee, di parole, di caratteri del file. con l'opzione -l (lines) conta le linee.

```
# wc < paradiso.text
4929 31624 182418
```

Dà, rispettivamente: il numero di righe nel file "paradiso.text" (4929), il numero di parole (31624), il numero di caratteri (182418).

Opzioni più significative:

-l (lines): dà il numero di linee del file

-w (words): dà il numero di parole del file; le parole sono separate

-c (characters): dà il numero dei caratteri compresi nel file.

sed (Stream Editor) fa "trasformazioni" su uno stream (flusso) di testo

Per eliminare dal file incasinato.html tutti i tag HTML di tipo "FONT" usiamo sed in questo modo:

```
# cat incasinato.html | sed 's/<FONT[^>]*>//g' > pulito.html
```

3.4 Cicli negli script

3.4.1 Cicli for

for <variabile> in <schema>

do

<comandi da eseguire in ciclo>

done

<schema> viene espanso come se fosse uno schema per l'individuazione del nome dei file. Questo genera una lista di stringhe. Gli elementi di questa lista vengono "assegnati", ad ogni iterazione del ciclo, alla <variabile>, che è una variabile d'ambiente.

Esempio:

```
for i in $*
do
   echo $i
done
```

questo script visualizza (echo) tutti gli argomenti che gli vengono passati, ognuno in una linea diversa.

Supponendo di avere editato i comandi precedenti in un file chiamato "script1", succede questo:

./script1 uno due 3 quattro 5

```
uno
due
3
quattro
5
```

Esempio:

```
for  append  in 01 03 05 10 zero; do
        echo 'contenuto del file ' file$append > file$append.txt
done
```

Questo script crea il file "file01.txt" scrivendoci dentro "contenuto del file file01", poi crea i file: "file03.txt", "file05.txt", "file10.txt", "filezero.txt", riempiendoli con una riga "personalizzata".

Cicli while

while <condizione>

do

<comandi eseguiti fino a quando la condizione non diviene vera>

done

La condizione può essere determinata anche eseguendo un comando. Se quel comando dà un risultato e non commette un errore il valore che genera è zero, che qui significa TRUE!

Cicli until

until <condizione>

do

<comandi eseguiti fino a quando la condizione non diviene falsa>

done

Istruzione condizionale (if)

if <condizione>

then

<comandi eseguiti se vera>

else

<comandi eseguiti se falsa>

fi

Esecuzione di casi diversi (case)

case <valore> in

<schema1>) <comandi da eseguire se schema1 è verificato> ;;

<schema2>) <comandi da eseguire se schema2 è verificato> ;;

<schema3>) <comandi da eseguire se schema3 è verificato> ;;

.. e così via

Esempio:

```
#       $# dà il numero di "parole" (parametri) passati dall'utente quando ha lanciato lo script
case $# in
   0) echo "non hai messo nessun parametro" ;;
   1) echo "hai messo un parametro" ;;
   2 | 4) echo "hai messo 2 o 4 parametri" ;;
   [5-7]) echo "hai messo 5, 6 o 7 parametri";;
   *) echo "hai messo 3 oppure più di 7 parametri" ;;
esac
```

Selezione di una opzione da parte dell'utente

select <variabile> in <schema>

do

<comandi da eseguire una volta sola>

done

<variabile> assume il valore di una delle stringhe che vengono espanse per <schema>, poi si esegue quello che c'è fra do e done.

Esempio:

bash: prova di select

select i in $*

do

echo "hai scelto $i"

done

se eseguito viene così:

$ ./script2 uno due 3 quattro 5

1) uno

2) due

3) 3

4) quattro

5) 5

#? 4

hai scelto quattro

Uno script per togliere i tag dai file HTML

Vogliamo fare uno script che prenda in ingresso il nome di un file ed una stringa "tag" e tolga tutti i tag "stringa" che sono nel file.

echo 'Eliminazione dei tag' $2 'nel file' $1 # 1

echo "Comando sed spedito: 's/<"$2"[^>]*>//g'" # 2

cat $1 | sed 's/<'$2'[^>]*>//g' > temp # 3

cp temp $1 # 4

rm temp # 5

cat $1 # 6

Il comando sed è analogo a quello già visto, al posto della stringa da sostituire viene usata la variabile $2, cioè il secondo parametro.

Il comando sed mette il suo risultato nel file temporaneo temp, non potrebbe farlo direttamente su $1, perchè altrimenti lo distruggerebbe (il file non è ancora chiuso quando come "sorgente" quando si apre come "destinazione")

Poi il file temp viene copiato nel file passato come parametro e cancellato.

Se il file dello script, reso eseguibile, si chiama EliminaTag, esso verrà usato in questo modo per togliere i tag <font ..>:

```
# EliminaTag FileDaPulire.htm font
```

Per togliere i tag invece bisogna fare così:

```
# EliminaTag FileDaPulire.htm
```

Da notare che il carattere "/" viene preceduto da due "\" (carattere escape dell'interprete dei comandi) non da uno solo. Infatti uno dei due "\" viene "intercettato" dall'interprete comandi e perchè giunga fino al comando sed bisogna metterne due. La linea 2 dello script fa vedere come questo accada in effetti.

4 Programmi CUI

4.0.1 vi

vi è un vecchio editor Unix che, per quanto datato, potrebbe essere utile quando non si usa XWindow e si lavora solo in linea di comando (meglio usarlo solo in "emergenza", quando non si ha altro)

4.0.2 mc (Midnight Commander)

è un clone per Linux dell'interfaccia utente di Norton Commander, programma di utilità per MSDOS. Permette il browsing fra i directory, il viewing e l'editing dei file.