Corso di Sistemi Operativi
A.A. 2008-2009
-
LA SHELL UNIX

Fabio Buttussi

Introduzione

- La parte del sistema operativo Unix dedita alla gestione dell'interazione con l'utente è la **shell**, ovvero, un'**interfaccia a carattere**:
  - l'utente impartisce i comandi al sistema digitandoli ad un apposito **prompt**;
  - il sistema stampa sullo schermo del terminale eventuali messaggi all'utente in seguito all'esecuzione dei comandi, facendo poi riapparire il prompt, in modo da continuare l'interazione.

- Versioni moderne di Unix forniscono **X-Windows**, un'interfaccia grafica (a finestre), che consente di inviare comandi tramite menu, utilizzando un mouse.

- **X-Term** è un emulatore di terminale che gira sotto X-Windows, fornendo localmente un'interfaccia a carattere. **Konsole** è un emulatore di terminale per KDE.

Tipi di Shell

sh	Bourne shell
bash	Bourne again shell
csh	C shell
tcsh	Teach C shell
ksh	Korn shell

Quando viene invocata una shell, automaticamente al login o esplicitamente:

1. viene letto un file speciale nella home directory dello user, contenente informazioni per l'inizializzazione;
2. viene visualizzato un **prompt**, in attesa che l'utente invii un comando;
3. se l'utente invia un comando, la shell lo esegue e ritorna al punto 2; ad esempio, `echo $SHELL` stampa sullo schermo del terminale il percorso della shell di login, mentre il comando `bash` invoca la shell bash.

Per terminare la shell si possono usare i seguenti metodi:

- premere Ctrl-D;
- digitare i comandi `logout` o `exit`.

Navigazione del filesystem

- Present working directory:
  `> pwd`
  `/home/bianchi`

- Change directory:
  `> cd /bin` (`cd` senza argomenti sposta l'utente nella sua home directory)

- Per spostarsi nella directory "madre":
  `> cd ..`
  dove `..` è l'alias per la directory "madre".

- `> pwd`
  `/home/bianchi`
  `> cd ./progetto` (dove `.` è l'alias per la directory corrente)
  `> pwd`
  `/home/bianchi/progetto`

Manipolazione di file e directory

- Listing dei file:
  `> ls`
  `> ls -l`
  `> ls -a`
  `> ls -al`
  `> ls -l /bin`
  `> ...`

- Creazione/rimozione di directory:
  `> mkdir d1`
  `> rmdir d1`

- Copia il file `f1` in `f2`:
  `> cp f1 f2`

- Sposta/rinomina il file `f1` in `f2`:
  `> mv f1 f2`

- `cp` e `mv` come primo argomento possono prendere una lista di file; in tal caso il secondo argomento deve essere una directory:
  `> cp f1 f2 f3 d1` (copia `f1`, `f2`, `f3` nella directory `d1`)

Un esempio d'uso del comando `ls`

Eseguendo il comando `ls -l /bin` si ottiene il seguente output:

```
...
lrwxrwxrwx    1 root     root            4 Dec  5  2000 awk -> gawk
-rwxr-xr-x    1 root     root         5780 Jul 13  2000 basename
-rwxr-xr-x    1 root     root       512540 Aug 22  2000 bash
...
```

da sinistra a destra abbiamo:

1. tipo di file (- file normale, `d` directory, `l` link, `b` block device, `c` character device),
2. permessi,
3. numero di hard link al file,
4. proprietario del file,
5. gruppo del proprietario del file,
6. grandezza del file in byte,
7. data di ultima modifica,
8. nome del file.

Il comando `chmod`

L'owner di un file può cambiarne i permessi tramite il comando `chmod`:

- `> chmod 744 f1` (imposta i permessi del file `f1` a `rwxr--r--`)
  Infatti: `rwxr--r--` $\rightsquigarrow$ `111 100 100` $\rightsquigarrow$ `7 4 4` (leggendo ogni gruppo in ottale)

- `> chmod u=rwx,go=r f1` (produce lo stesso effetto del comando precedente)
  dove `u` rappresenta l'owner, `g` il gruppo e `o` il resto degli utenti (world)
  Inoltre:
  `+` aggiunge i permessi che lo seguono,
  `-` toglie i permessi che lo seguono,
  `=` imposta esattamente i permessi che lo seguono.
  Quindi l'effetto di `chmod g+r f1` è in generale diverso da `chmod g=r f1`.

Link e link simbolici

- Creazione di **link (hard)**:
  `> ln f2 f2_new`
  il file f2_new ha lo stesso inode di f2
  `> ln f1 g1`

- Creazione di un **link simbolico**:
  `> ln -s g1 g1_new`
  un link simbolico è un tipo di file speciale in Unix; g1_new è un file di testo che contiene il pathname di g1

Visualizzazione file e manuale

- Visualizzazione del contenuto di un file:

  ```
  > cat f1
  > more f1
  > less f1
  > tail f1
  > head f1
  ```

- Consultazione del manuale on-line:

  * Sezione 1 : comandi
  * Sezione 2 : system call
  * Sezione 3 : funzioni di libreria

  ...

  ```
  > man passwd
  > man -a passwd
  > man -s2 mkdir
  > man man
  ```

l'Editor `vi`

- Negli ambienti Unix esistono molti editor di testo diversi; tuttavia `vi` è l'unico che siamo sicuri di trovare in qualsiasi variante di Unix.

- `vi` (**visual editor**) è stato scritto per essere utilizzabile con qualsiasi tipo di terminale.

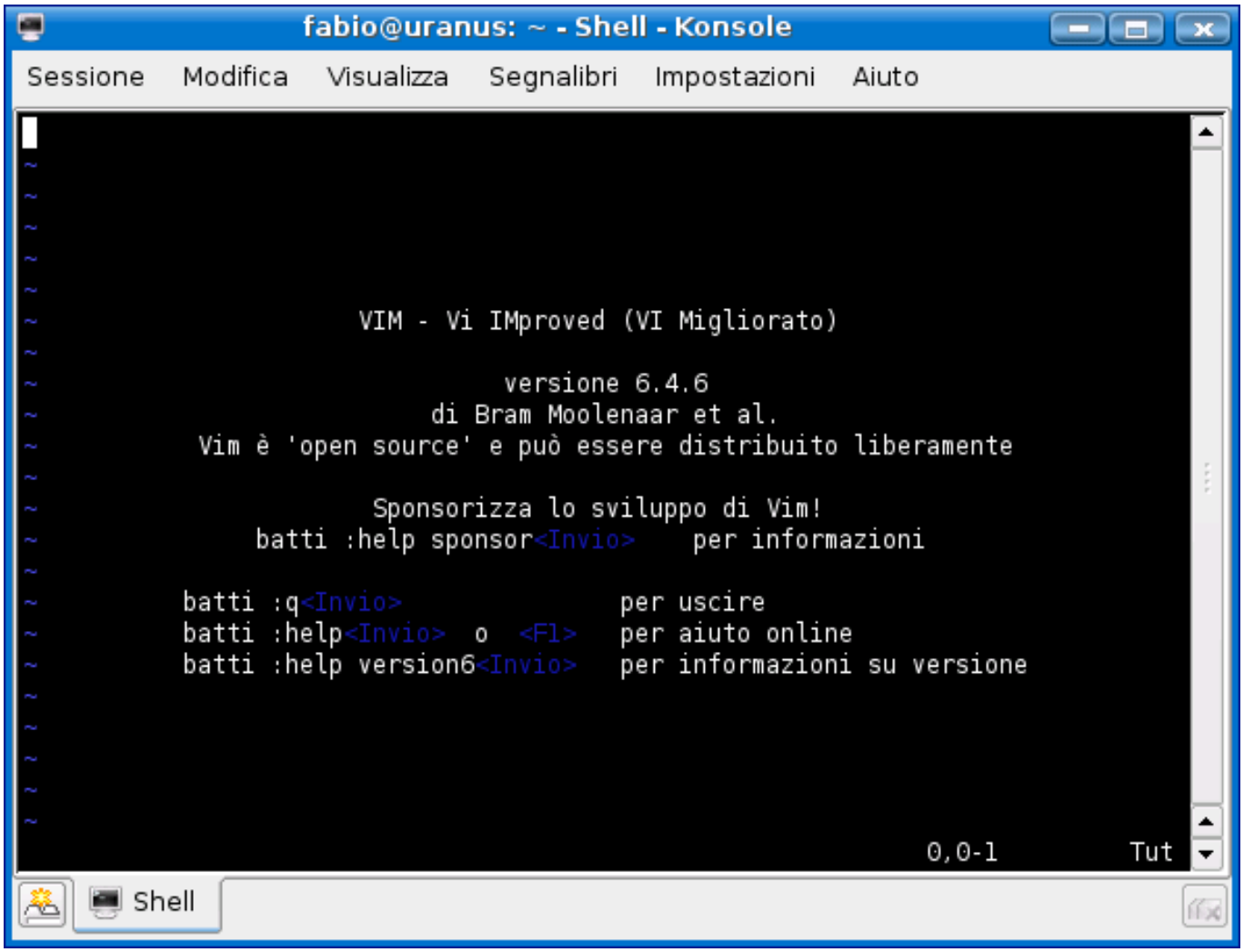
- `$ vi filename`

  invoca `vi` aprendo il file `filename` (se non esiste, viene creato).

- `vi` ha tre **modalità**:

  1. **edit** mode (all'avvio di `vi` si è in questa modalità),
  2. **insert** mode,
  3. **command** mode.

Interfaccia di `vi`



L'editor Emacs/XEmacs

Emacs, E(ditor) Mac(ro)s, è un applicativo che non fa parte di Unix; è stato scritto da R. Stallman nel 1975. Digitando al prompt

```
$ emacs <file>
```

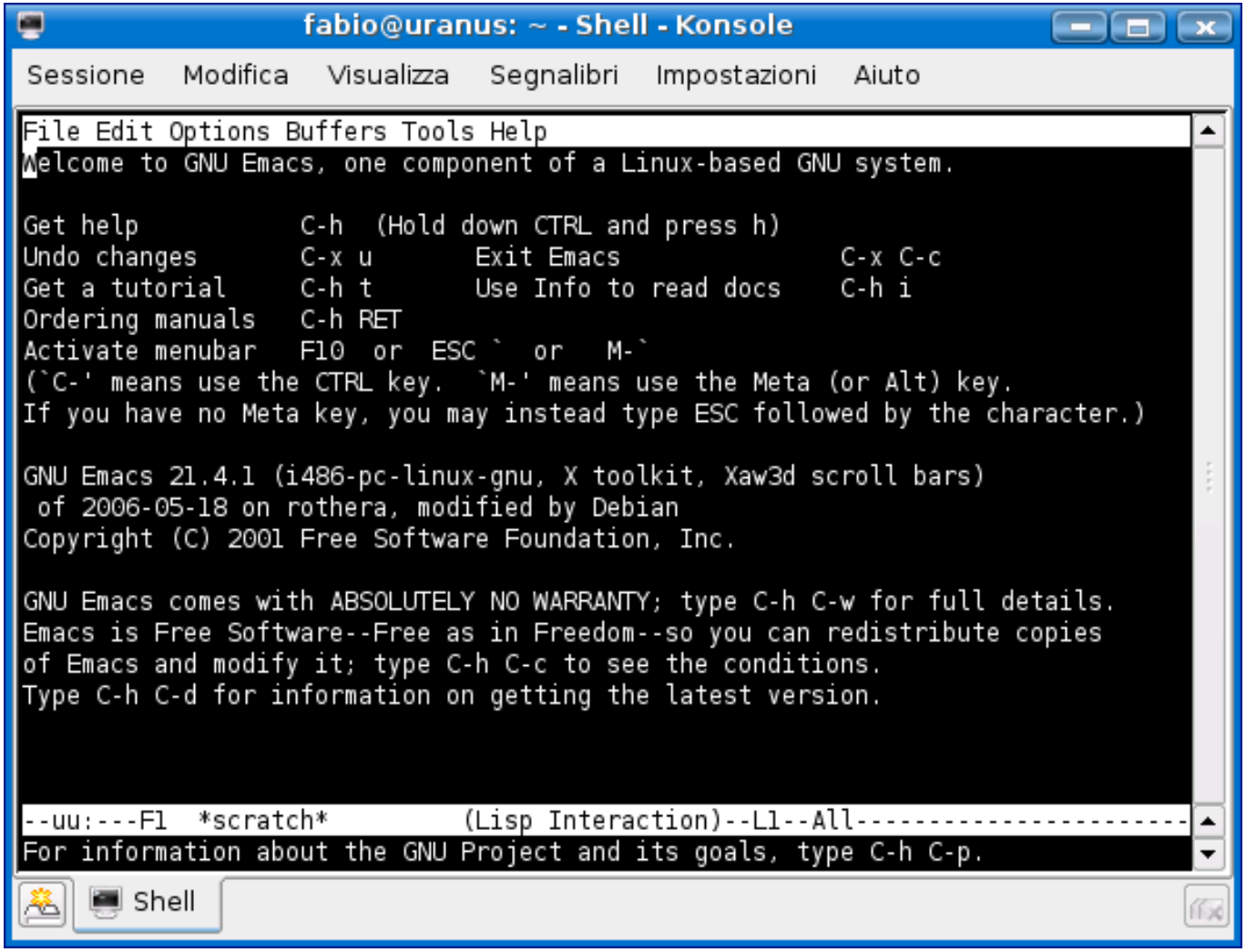
si apre una finestra in cui viene visualizzato il contenuto del file `<file>`, che può essere editato normalmente. Tale contenuto è memorizzato in un buffer (struttura dati interna di Emacs).

Ci sono due modi per accedere ai comandi di editing:

1. barra dei menu,

2. **caratteri di controllo**.

Interfaccia di `emacs`



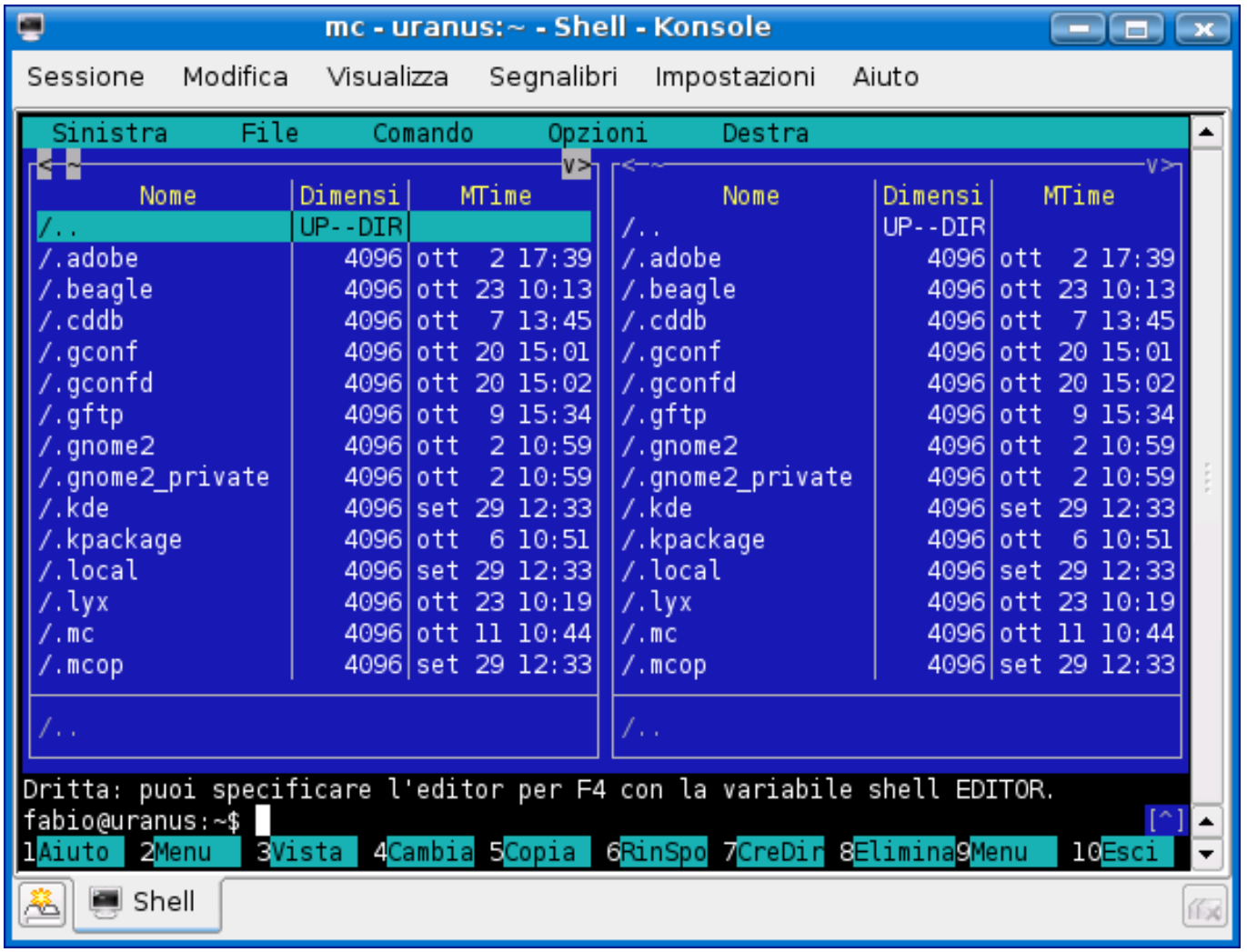
L'alternativa `mc` (Midnight Commander)

Midnight Commander, che si esegue con `mc`, un browser per l'esplorazione del filesystem da console, che permette anche di visualizzare e modificare i file.

Pur non utilizzando X, fornisce una interfaccia pi user-friendly rispetto ad altri editor e permette di:

- navigare tra file e cartelle utilizzando le frecce e il tasto `Invio`;
- visualizzare un file con `F3`;
- modificare un file con `F4`;
- salvare un file con `F2` (in edit mode);
- uscire dall'editor e da `mc` con `F10`.

Interfaccia di `mc`



Ulteriori comandi sui file

- Confronto tra file:

  1. `> cmp file1 file2`
     restituisce il primo byte ed il numero di linea in cui `file1` e `file2` differiscono (se sono uguali, non viene stampato nulla a video).
  2. `> diff file1 file2`
     restituisce la lista di cambiamenti da apportare a `file1` per renderlo uguale a `file2`.

- Ricerca di file:
  `> find` <*pathnames*> <*expression*>
  attraversa ricorsivamente le directory specificate in <*pathnames*> applicando le regole specificate in <*expression*> a tutti i file e sottodirectory trovati.
  <*expression*> può essere una fra le seguenti:
  1. opzione,
  2. condizione,
  3. azione.

Esempi d'uso di `find`

- `> find . -name '*.c' -print`
  cerca ricorsivamente a partire dalla directory corrente tutti i file con estensione `c` e li stampa a video.

- `> find . -name '*.bak' -ls -exec rm {} \;`
  cerca ricorsivamente a partire dalla directory corrente tutti i file con estensione `bak`, li stampa a video con i relativi attributi (`-ls`) e li cancella (`-exec rm {} \;` Il carattere `\` serve per fare il "quote" del `;`).

- `> find /etc -type d -print`
  cerca ricorsivamente a partire dalla directory `/etc` tutte e solo le sottodirectory, stampandole a video.

I Metacaratteri della Shell Unix

La shell Unix riconosce alcuni caratteri speciali, chiamati **metacaratteri**, che possono comparire nei comandi.

Quando l'utente invia un comando, la shell lo scandisce alla ricerca di eventuali metacaratteri, che processa in modo speciale.
Una volta processati tutti i metacaratteri, viene eseguito il comando.

Esempio:

```
user> ls *.java

 Albero.java          div.java       ProvaAlbero.java
 AreaTriangolo.java   EasyIn.java    ProvaAlbero1.java
 AreaTriangolo1.java  IntQueue.java
```

Il **metacarattere** `*` all'interno di un pathname è un'**abbreviazione** per un nome di file. Il pathname `*.java` viene espanso dalla shell con tutti i nomi di file che terminano con l'estensione `.java`. Il comando `ls` fornisce quindi la lista di tutti e soli i file con tale estensione.

Abbreviazione del Pahtname

I seguenti metacaratteri, chiamati **wildcard** sono usati per **abbreviare** il nome di un file in un pathname:

Metacarattere	**Significato**
*	stringa di 0 o più caratteri
?	singolo carattere
[ ]	singolo carattere tra quelli elencati
{ }	stringa tra quelle elencate

Esempi:

```
user> cp /JAVA/Area*.java /JAVA_backup
```

copia tutti i files il cui nome inizia con la stringa `Area` e termina con l'estensione `.java` nella directory `JAVA_backup`.

```
user> ls /dev/tty?
/dev/ttya /dev/ttyb
```

Il "quoting"

Il meccanismo del **quoting** è utilizzato per inibire l'effetto dei metacaratteri. I metacaratteri a cui è applicato il quoting perdono il loro significato speciale e la shell li tratta come caratteri ordinari.
Ci sono tre meccanismi di quoting:

- il metacarattere di **escape** \ inibisce l'effetto speciale del metacarattere che lo segue:

```
user> cp file file\?
user> ls file*
file     file?
```

- tutti i metacaratteri presenti in una stringa racchiusa tra **singoli apici** perdono l'effetto speciale:

```
user> cat 'file*?'
...
```

- i metacaratteri per l'abbreviazione del pathname presenti in una stringa racchiusa tra **doppi apici** perdono l'effetto speciale (ma non tutti i metacaratteri della shell):

```
user> cat "file*?"
...
```

Ridirezione dell'I/O

Di default i comandi Unix prendono l'input da **tastiera** (**standard input**) e mandano l'**output** ed eventuali **messaggi di errore** su video (**standard output, error**).

L'input/output in Unix può essere **rediretto** da/verso **file**, utilizzando opportuni metacaratteri:

Metacarattere	**Significato**
>	ridirezione dell'output
>>	ridirezione dell'output (append)
<	ridirezione dell'input
<<	ridirezione dell'input dalla linea di comando ("here document")
2>	ridirezione dei messaggi di errore (bash Linux)

Esempi:

```
user> ls LABSO > temp
user> more temp
lezione1.aux lezione1.log lezione1.tex lezione2.dvi lezione2.tex
lezione1.dvi lezione1.ps lezione2.aux lezione2.log lezione2.tex
```

Altri esempi

```
user> echo ciao a tutti >file     # ridirezione dell'output
user> more file
ciao a tutti

user> echo ciao a tutti >>file      # ridirezione dell'output (append)
user> more file
ciao a tutti
ciao a tutti
```

Il comando `wc` (**word counter**) fornisce numero di linee, parole, caratteri di un file:

```
user> wc <progetto.txt
21 42 77

user> wc <<delim      # here document
?  queste linee formano il contenuto
?  del testo
?  delim
2    7    44

user> man -s2 passwd      # ridirezione dei messaggi di errore
No entry for passwd in section(s) 2 of the manual.
user> man -s2 passwd 2>temp
```

Pipe

Il metacarattere | (**pipe**) serve per comporre n comandi "in cascata" in modo che l'output di ciascuno sia fornito in input al successivo. L'output dell'ultimo comando è l'output della pipeline.

La sequenza di comandi

```
user> ls /usr/bin > temp
user> wc -w temp
459
```

ha lo stesso effetto della pipeline:

```
user> ls /usr/bin | wc -w
459
```

I comandi `ls` e `wc` sono eseguiti in parallelo: l'output di `ls` è letto da `wc` mano a mano che viene prodotto.

Per mandare in stampa la lista dei files in `/usr/bin`:

```
user> ls /usr/bin | lpr
```

Per visualizzare l'output di `ls` pagina per pagina

```
user> ls | more
```

Bash: command line editing

La shell bash mette a disposizione dell'utente dei semplici **comandi di editing** per facilitare la ripetizione degli eventi:

- utilizzando i **tasti cursore**:
  - con la **freccia verso l'alto** si scorre l'history list a ritroso (un passo alla volta) facendo apparire al prompt il comando corrispondente all'evento;
  - analogamente con la **freccia verso il basso** si scorre l'history list nella direzione degli eventi più recenti.
  - le frecce sinistra e destra consentono di spostare il cursore sulla linea di comando verso il punto che si vuole editare;

- le combinazioni di tasti `Ctrl-A` e `Ctrl-E` spostano il cursore, rispettivamente all'inizio ed alla fine della linea di comando;

- il tasto `Backspace` consente di cancellare il carattere alla sinistra del cursore;

- il tasto invio (`enter`) esegue il comando.

Bash: command completion (I)

Una caratteristica molto utile della shell bash è la sua abilità di **tentare di completare** ciò che stiamo digitando al prompt dei comandi (nel seguito `<Tab>` indica la pressione del tasto `Tab`).

```
$ pass<Tab>
```

La pressione del tasto `<Tab>` fa in modo che la shell, sapendo che vogliamo impartire un comando, cerchi quelli che iniziano con la stringa `pass`. Siccome l'unica scelta possibile è data da `passwd`, questo sarà il comando che ritroveremo automaticamente nel prompt.

Se il numero di caratteri digitati è insufficiente per la shell al fine di determinare univocamente il comando, avviene quanto segue:

- viene prodotto un suono di avvertimento al momento della pressione del tasto `Tab`;
- alla seconda pressione del tasto `Tab` la shell visualizza una lista delle possibili alternative.
- digitando ulteriori caratteri, alla successiva pressione del tasto `Tab`, la lunghezza della lista diminuirà fino ad individuare un unico comando.

Bash: command completion (II)

Oltre a poter completare i comandi, la shell bash può anche completare i nomi dei file usati come argomento:

```
$ tail -2 /etc/p<Tab><Tab>
passwd  printcap  profile
$tail -2 /etc/pa<Tab><Tab>
bianchi:fjKppCZxEvouc:500:500::/home/bianchi:/bin/bash
rossi:Yt1a4ffkGr02:501:500::/home/rossi:/bin/bash
```

In questo caso alla prima doppia pressione del tasto `Tab`, la shell presenta tre possibili alternative; digitando una `a` e premendo il tasto `Tab`, la shell ha una quantità di informazione sufficiente per determinare in modo univoco il completamento del nome di file.

Controllo dello spazio su disco

Per controllare la quantità di spazio su disco in uso:

```
user> df
  /                (/dev/dsk/c0t0d0s0 ): 2231020 blocks   304297 files
  /proc            (/proc             ):       0 blocks    11692 files
  /opt             (/dev/dsk/c0t0d0s3 ): 2486076 blocks   325130 files
  /usr/local       (/dev/dsk/c0t0d0s7 ): 4067088 blocks   430227 files
  /opt/solaris2    (apphost:/opt/solaris2): 1995968 blocks   286013 files
```

Legenda: il primo campo contiene il nome del file system; il secondo il device corripondente (eventualmente virtuale); il terzo il numero di blocchi occupati; il quarto il numero di inode.

Per controllare la quantità di spazio su disco utilizzata da una directory (in blocchi):

```
user> du LABORATORIO_SO

  8       LABORATORIO_SO/LABSO/CVS
  16884   LABORATORIO_SO/LABSO
  16      LABORATORIO_SO/scriptColonne
  14      LABORATORIO_SO/linguaggio_c
  17342   LABORATORIO_SO
```

Controllo di processi

Ogni processo del sistema ha un **PID** (**Process Identity Number**).
Ogni processo può generare nuovi processi (figli).
La radice della gerarchia di processi è il processo **init** con PID=1.
init è il primo processo che parte al boot di sistema.

Il comando `ps` fornisce i processi presenti nel sistema:

```
user> ps   # fornisce i processi dell'utente associati al terminale corrente

   PID          TTY          TIME     CMD
 23228          pts/15       0:00     xdvi.bin
  9796          pts/15       0:01     bash
 23216          pts/15       0:04     xemacs-2
  9547          pts/15       0:00     csh
```

Legenda: `PID` = PID; `TTY` = terminale (virtuale); `TIME` = tempo di CPU utilizzato; `CMD` = comando che ha generato il processo.

Per ottenere il nome del terminale corrente:

```
user> tty
 /dev/pts/15
```

Il comando `ps` e sue varianti (I)

Per ottenere tutti i processi nel sistema associati ad un terminale (`-a`), full listing (`-f`):

```
user> ps -af

   UID   PID  PPID  C     STIME  TTY      TIME CMD
lenisa 10922  9560  0    Oct 17  pts/17   0:00 bash
pietro 23410 23409  0  11:07:08  pts/26   0:01 xdvi.bin -name xdvi main.dvi
  root 24188  9807  0  12:34:10  pts/13   0:00 ps -af
         ....
```

Legenda: `UID` = User Identifier; `PPID` = Parent PID; `C` = informazione obsoleta sullo scheduling; `STIME` = data di inizio del processo.

Il comando `ps` e sue varianti (II)

Per ottenere tutti i processi nel sistema, anche non associati ad un terminale (`-e`), long listing (`-l`):

```
user> ps -el
```

```
 F S   UID   PID  PPID  C PRI NI    ADDR     SZ   WCHAN  TTY      TIME CMD
19 T     0     0     0  0   0 SY       ?      0          ?        0:12 sched
 8 S     0     1     0  0  41 20       ?    100        ?  ?        0:03 init
 8 S   140 12999 12997  0  56 20       ?    278        ?  pts/12   0:00 tcsh
 8 R   159  9563  9446  0  50 20       ?   2110           ?        0:30 acroread
        ....
```

Legenda: `F` = flag obsoleti; `S` = stato del processo (T=stopped); `PRI` = priorità; `NI` = nice value (usato per modificare la priorità); `ADDR` = indirizzo in memoria; `SZ` = memoria virtuale usata; `WCHAN` = evento su cui il processo è sleeping.

Terminazione di un processo

Per arrestare un processo in esecuzione si può utilizzare

- la sequenza `Ctrl-c` dal terminale stesso su cui il processo è in esecuzione;
- il comando `kill` seguito dal `PID` del processo (da qualsiasi terminale):

```
user> ps

  PID    TTY      TIME CMD
      .......
  28015  pts/14   0:01 xemacs
      .......

user> kill 28015
```

- il comando `kill` con il segnale SIGKILL

```
user> kill -9 28015

user> kill -s kill 28015
```

Processi in background

Un comando (pipeline, sequenza) seguito da `&` dà luogo ad uno o più **processi in background**. I processi in background sono eseguiti in una **sottoshell**, **in parallelo** al processo padre (la shell) e **non** sono controllati da tastiera.
I processi in background sono quindi utili per eseguire task in parallelo che non richiedono controllo da tastiera.

```
user> xemacs &
  [1] 24760
```

[1] è il numero del job, 24760 il PID del processo

```
user> xemacs &
user> ls -R / >temp 2>err &
```

Il comando `jobs` mostra la lista dei job in esecuzione:

```
user> jobs
  [1]   Running                 xemacs &
  [2]-  Running                 xemacs &
  [3]+  Running                 ls -R / >tmp 2>err
```

Controllo di job

Un job si può **sospendere** e poi **rimandare in esecuzione**

```
user> cat >temp    # job in foreground

   Ctrl-z   # sospende il job

  [1]+ Stopped

user> jobs
  [1]+ Stopped     cat >temp

user> fg    # fa il resume del job in foreground

   Ctrl-z   # sospende il job

user> bg    # fa il resume del job in background

user> kill %1   # termina il job 1
  [1]+ Terminated
```

Schema riassuntivo shell Bash

Argomento	**Alcuni comandi utili**
Shell e sessione	sh, bash, csh, tcsh, ksh, logout, exit, history, alias, unalias, man, who, date
File e directory	pwd, cd, ls, mkdir, rmdir, cp, mv, rm, ln, chmod, touch, mount, df, du, basename, cmp, diff, find
Processi e job	ps, top, kill, jobs, fg, bg
Visualizzazione	echo, cat, more, less, tail, head
Filtro	wc, uniq, grep, fgrep, egrep, sort, tr, cut, paste
Editor	vi, emacs, xemacs, mc, sed, awk
Script	set, shift, if then else fi, while do done, until do done, for in do done, case in esac

Metacaratteri comuni a tutte le shell (I)

Simbolo	Significato	Esempio d'uso
`>`	Ridirezione dell'output	`ls >temp`
`>>`	Ridirezione dell'output (append)	`ls >>temp`
`<`	Ridirezione dell'input	`wc -l <text`
`<<delim`	ridirezione dell'input da linea di comando (here document)	`wc -l <<delim`
`*`	Wildcard: stringa di 0 o più caratteri, ad eccezione del punto (.)	`ls *.c`
`?`	Wildcard: un singolo carattere, ad eccezione del punto (.)	`ls ?.c`
`[...]`	Wildcard: un singolo carattere tra quelli elencati	`ls [a-zA-Z].bak`
`{...}`	Wildcard: le stringhe specificate all'interno delle parentesi	`ls {prog,doc}*.txt`

Metacaratteri comuni a tutte le shell (II)

Simbolo	Significato	Esempio d'uso
`|`	Pipe	`ls | more`
`;`	Sequenza di comandi	`pwd;ls;cd`
`||`	Esecuzione condizionale. Esegue un comando se il precedente fallisce.	`cc prog.c || echo errore`
`&&`	Esecuzione condizionale. Esegue un comando se il precedente termina con successo.	`cc prog.c && a.out`
`(...)`	Raggruppamento di comandi	`(date;ls;pwd)>out.txt`
`#`	Introduce un commento	`ls # lista di file`
`\`	Fa in modo che la shell non interpreti in modo speciale il carattere che segue.	`ls file.\*`
`!`	Ripetizione di comandi memorizzati nell'history list	`!ls`