

# Fondamenti di Informatica

## Comandi UNIX

Tullio Facchinetti

`<tullio.facchinetti@unipv.it>`

1 marzo 2016

13:51

`http://robot.unipv.it/toolleeo`

la storia del linguaggio C è strettamente legata a quella di UNIX – e oggi giorno di Linux

- negli anni '70 il C è nato per sviluppare programmi di sistema e driver per UNIX
- il kernel di Linux è implementato in C

## Autenticazione (login)

- l'autenticazione è necessaria per l'accesso al sistema
- permette il collegamento di più utenti, contemporaneamente o in momenti diversi
- è così possibile individuare univocamente l'identità dell'utente
  - permette di gestire opportunamente i permessi di accesso a file e directory
  - storicamente questa informazione veniva usata per contabilizzare il tempo di utilizzo del calcolatore, che era una risorsa molto limitata, per farlo poi pagare

## Autenticazione (login)

- tipica accoppiata username/password che devono essere forniti al sistema
- il processo di autenticazione inizia scrivendo il *nome utente*, detto anche *username*
- lo username viene assegnato dall'amministratore del sistema
- lo username identifica univocamente l'utente
- in seguito vanno digitati i caratteri della password
- i caratteri battuti come password sono tipicamente occultati, cioè non vengono visualizzati sul video
- questo evita che la password possa essere “rubata” da malintenzionati eventualmente appostati alle spalle dell'utente che sta effettuando l'autenticazione

## Autenticazione (login): esempio

```
login: utente007
Password: pl67kf2
user007@europa:~$
```

- la schermata di login può variare a seconda del sistema
- spesso si tratta di una schermata grafica con pulsanti, loghi e immagini
- nell'esempio la password viene mostrata per esigenze didattiche; in realtà questo non accade
- se l'accesso ha successo, viene presentato il prompt formato, in questo caso, dai caratteri `user007@europa:~$`

## Autenticazione (login)

- errori su username o password vengono opportunamente notificati
- la richiesta di login viene ripresentata finché l'autenticazione non ha successo

```
login: utente1
```

```
Password:
```

```
Login incorrect
```

```
login:
```

l'username `root` è associato ad un utente speciale con funzioni di amministrazione e gestione del sistema; molte funzioni sono accessibili soltanto a tale utente

username e password sono **case-sensitive**

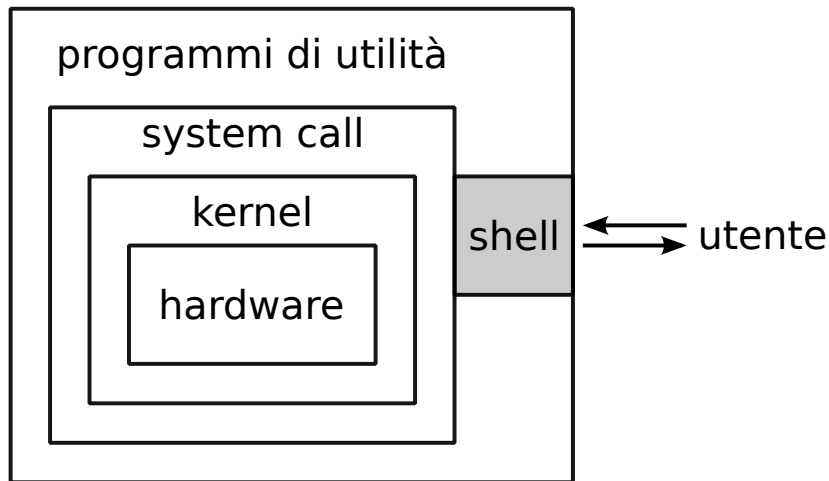
- il *case* di un carattere indica il fatto che esso sia maiuscolo o minuscolo
- un sistema si dice *case sensitive* se esso è sensibile al *case*, ovvero se viene fatta distinzione tra caratteri maiuscoli e minuscoli
- pertanto, per esempio, le parole `user007`, `USER007` e `User007` sono considerate tutte diverse tra loro

la shell è un *interprete* che riceve i comandi e ne esegue le operazioni associate

- è così chiamata in quanto costituisce lo strato più esterno del sistema, cioè quello più vicino all'utente
- UNIX può infatti essere pensato come una serie di livelli logici posti tra la macchina e l'utente, il più esterno dei quali è appunto la shell



## L'interprete dei comandi: la shell



## L'interprete dei comandi: la shell

- l'interazione tra l'utente e l'interprete avviene per mezzo della cosiddetta *linea di comando*, nella quale l'utente introduce i comandi sotto forma di *stringhe di testo*, ovvero sequenze di caratteri
- i caratteri inseriti dall'utente sono riprodotti sullo schermo accanto al prompt
- la riga di comando viene presa in considerazione dalla shell solo quando viene premuto il tasto Invio/Enter/Return
- fino ad allora, il suo contenuto si trova in un'area provvisoria detta *buffer di input*, e può essere corretto con il tasto di cancellazione

## L'interprete dei comandi: la shell

- esistono vari programmi che implementano la shell, tutti più o meno compatibili tra loro in termini di sintassi
- i programmi di shell più noti e utilizzati sono `ksh` (Korn shell), `csh` (C shell) e `bash` (Bourne-Again SHell)

al seguente indirizzo si può trovare il **Bash Reference Manual** una trattazione molto completa della shell Bash:

<http://www.gnu.org/software/bash/manual/bashref.html>

## L'interprete dei comandi: la shell

- il *prompt* è il segnale con cui la shell si mostra pronta a ricevere altro input dopo aver elaborato l'input precedente
- il prompt può essere personalizzato per mostrare informazioni quali lo username, il nome della macchina che si sta utilizzando (utile quando ci si collega a computer remoti), e la directory corrente

esempio:

```
user1@europa:~$
```

## L'interprete dei comandi: la shell

il quale può essere scomposto nei seguenti elementi:

```
user1 @ europa : ~ $
```

`user1` è lo username

`@` è un carattere convenzionale (si legge "at") che introduce il nome del computer al quale si è collegati

`europa` è il nome del computer in uso

`:` è un carattere convenzionale che introduce il percorso della directory corrente

`~` è la directory corrente

`$` è un carattere convenzionale per delimitare il prompt dall'input dell'utente

## Sintassi dei comandi UNIX

la sintassi dei comandi UNIX è la seguente

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

dove:

- `comando` indica l'operazione da compiere
- `[opzioni]` permette di specificare delle varianti al comando
- `[argomenti]` specifica i parametri del comando

le parentesi quadre non sono parte del comando che deve essere impartito; servono ad indicare che opzioni e argomenti sono dei parametri opzionali, cioè possono essere omessi

## Sintassi dei comandi UNIX

esempi di comandi sono i seguenti:

```
user1@europa:~$ ls
```

```
user1@europa:~$ cat file_da_visualizzare
```

```
user1@europa:~$ cp file_da_copiare nuovo_file
```

```
user1@europa:~$ mv file_da_rinominare file_con_nuovo_nome
```

```
user1@europa:~$ mkdir nuova_directory
```

```
user1@europa:~$ echo visualizza questa stringa a video
```

- il *file system* è una struttura utilizzata per memorizzare e organizzare le informazioni in un sistema di calcolo
- i dati sono immagazzinati all'interno di *file*, che sono costituiti quindi dal blocco di informazioni che si intende memorizzare
- i file non contengono soltanto dati, ma possono contenere le istruzioni da eseguire corrispondenti ad un determinato programma (i cosiddetti programmi eseguibili)



## I nomi dei file

- ad ogni file è associato un nome
- il nome può essere composto dai seguenti caratteri:

A...Z a...z 0...9 \_ - . ,

- cambiando sistema operativo cambia anche l'insieme dei caratteri validi per un nome di file
- i caratteri sopra riportati sono però validi per pressoché ogni sistema operativo, e quindi in particolare anche per UNIX

i sistemi di tipo UNIX distinguono tra lettere maiuscole e minuscole (sono *case-sensitive*)

- alcuni esempi di nomi di file sono: `lezione`, `lezione.doc`, `LEZIONE.doc`, `Lezione.doc`, `lezione.old`, `file_mio`, `19.nov.92`
- questi nomi di file rappresentano tutti file diversi tra loro

è tipico che il nome di un file sia costruito nella forma  
`nome.estensione`

- `nome` rappresenta il nome vero e proprio del file che ne identifica il contenuto
- `estensione` è una sequenza di caratteri che indica il tipo di dati contenuti del file
- nome ed estensione sono tipicamente separati dal punto

## Estensioni dei file

- l'uso dell'estensione è non indispensabile
- è però utile per comprendere immediatamente la tipologia del contenuto del file

### esempi

- le estensioni `.jpg`, `.gif`, `.png` o `.svg` contengono immagini in vari formati con caratteristiche diverse
- le estensioni `.c`, `.cpp`, `.java`, `.py` e `.m` sono utilizzate per i file sorgente di programmi scritti rispettivamente in linguaggio C, C++, Java, Python e Matlab
- le estensioni `.doc`, `.odt`, `.xls` e `.ppt` sono usate da programmi da ufficio

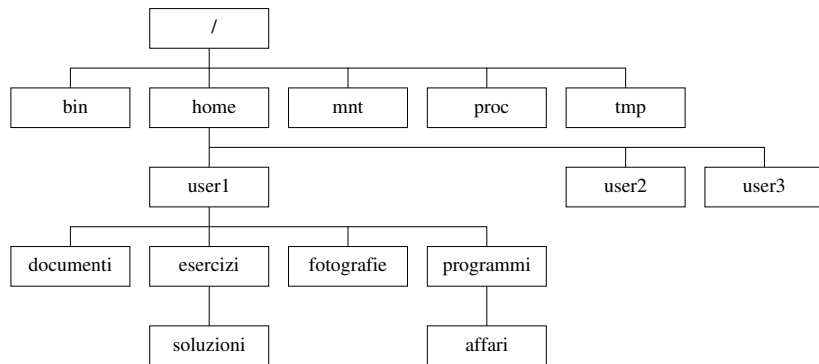
## Estensioni dei file

- esistono migliaia di estensioni diverse
- in alcune situazioni, la stessa estensione ha significati diversi, anche se questi sono casi molto particolari, poco comuni e soprattutto non interessano le più comuni estensioni di file
- due file indentici per quanto riguarda il contenuto potrebbero avere estensioni diverse
- l'uso non convenzionale delle estensioni ha il solo risultato di confondere le idee
- il carattere punto non necessariamente viene soltanto utilizzato solo per introdurre l'estensione
- per esempio, il nome `archivio.tar.gz` è lecito e indica un file archiviato col comando `tar` di UNIX e compresso col programma `gzip`

le *directory* servono per organizzare in modo gerarchico i file, e costituiscono il file system

- ogni file ha una directory che lo contiene
- la struttura delle directory è quella di un *albero*
- ogni directory ha un solo genitore, in cui è contenuta e di cui si dice figlia
- la struttura delle directory si dice gerarchica perché la relazione genitore-figlio determina una gerarchia
- di norma una qualsiasi directory (es. `user1`) si trova dentro un'altra directory

## Le directory: esempio di albero delle directory



esiste una directory che non è contenuta in nessun'altra: si chiama `root` (radice dell'albero) e si indica con la barra `/`

- la barra che indica la directory `root` si chiama *slash*
- ATTENZIONE: non è da confondersi con la barra `\`, la quale è chiamata *backslash*



ad ogni istante è definita una **directory corrente**  
o **directory di lavoro**

- la directory corrente è quella nella quale “ci si trova” in un determinato momento durante l’uso della macchina
- come vedremo, la directory corrente può essere cambiata in caso di necessità
- ...ovvero, ci si può **spostare** nell’albero delle directory

## Percorsi o pathname

- directory o file diversi **possono avere lo stesso nome**, ma solo qualora siano **contenuti in directory diverse**
- pertanto, file e directory, intesi come nodi di un albero, **non possono essere individuati univocamente soltanto dal nome**

ad ogni file o directory è associato un **percorso** o **pathname**

esempio:

il file `immagine01.jpg` può essere contenuto della directory `fotografie` oppure `documenti`

si consideri il file o directory di nome `fname`

### percorso assoluto

- 1 localizza `fname` sull'albero *rispetto alla directory root*
- 2 è identificato dalle directory da `/` fino a `fname` compreso, separati da `/`

esempio:

```
/home/user1/esercizi/soluzioni/fname
```

- dipende solo dalla posizione del file nel file-system
- è unico

si consideri il file o directory di nome `fname`

### percorso relativo

- 1 localizza `fname` *rispetto alla directory corrente*
- 2 è dato dalle directory a partire da quella corrente (esclusa) fino a `fname` compresa, separati da `/`

esempio:

`esercizi/soluzioni/fname`

- dipende dalla directory corrente
- in questo caso, una directory corrente valida è `/home/user1/`

alcune caratteristiche dei percorsi relativi e assoluti:

- un percorso assoluto inizia per /, altrimenti è relativo
- in alcun caso un percorso relativo NON inizia con lo slash
- nella scrittura di un percorso, il carattere / ha 2 usi:
  - 1 indica la directory `root` se posto all'inizio come primo carattere del pathname
  - 2 funge da separatore di directory se posto in mezzo al pathname

Il comando `pwd`: visualizzazione della directory corrente

il comando `pwd` (*(P)rint (W)orking (D)irectory*) stampa il percorso assoluto della directory corrente

ad esempio, se ci si trova nella directory `programmi` in `user1`, si ottiene il seguente risultato:

```
user1@europa:~/programmi$ pwd
/home/user1/programmi
```

- il comando `pwd` non è implementato come un programma presente sul disco e chiamato `pwd`
- è un comando interno alla shell
- questo perché la directory corrente è un attributo associato al processo corrente, ed è quindi diversa per ogni processo

## Elencare il contenuto di una directory

il comando `ls` (*list*) mostra il contenuto della directory corrente

il formato completo di `ls` è:

```
ls [opzioni] [lista di file o dir]
```

- le parentesi quadre indicano che il contenuto delle parentesi stesse è opzionale, cioè è possibile ometterlo nell'invocazione del comando senza pregiudicarne l'effetto
- l'aggiunta di una o più opzioni può essere fatta per modificare leggermente il comportamento del comando

## Le opzioni più utili del comando `ls`

Le opzioni del comando `ls` più comunemente utilizzate sono:

`ls -a` elenca anche i file (normalmente invisibili) il cui nome comincia per `.` (punto)

`ls -l` elenca in formato lungo

`ls -t` elenca a partire dal file più recente

`ls -C` visualizza l'elenco incolonnato

`ls -R` (maiuscolo) elenca ricorsivamente anche le sotto-directory

`ls -r` (minuscolo) elenca i file in ordine inverso sotto-directory

`ls -f` non ordina i file, abilita l'opzione `-a`, disabilita la `-l` e i colori



## Esempio di uso del comando `ls`

un esempio di utilizzo del comando `ls` è il seguente:

```
user1@europa:$ ls -l
-rw-r--r-- 1 toolleeo root 3500 2009-06-26 15:39 programma.c
```

- si richiede di visualizzare il contenuto della directory corrente utilizzando il formato lungo
- contiene il solo file `programma.c`

## Esempio di uso del comando `ls`

il formato lungo visualizza una serie di utili informazioni riguardo ai file:

- `-rw-r--r--` specifica il tipo di file ed i permessi di accesso al file
- `1` indica il numero di cosiddetti “hard links” collegati al file
- `toolleeo` è l'utente che possiede il file
- `root` è il gruppo associato al file
- `3500` è la dimensione del file in byte
- `2009-06-26` e `15:39` sono la data e l'ora di ultima modifica del file
- `programma.c` è il nome del file

## Il comando `which`

- l'esecuzione di un comando da parte della shell comporta tipicamente l'esecuzione di un programma memorizzato su disco
- il programma da eseguire ha lo stesso nome del comando impartito
- il programma è memorizzato in una **directory di sistema**, ovvero una directory standard nella quale la shell si aspetta siano memorizzati i programmi associati ai comandi

`which` stampa il percorso assoluto del programma che viene eseguito quando viene impartito un determinato comando

## Il comando `which`

esempio:

```
$ which ls  
/bin/ls
```

è possibile verificare che effettivamente il programma `ls` è un file presente su disco utilizzando il comando `ls` stesso:

```
$ ls -l /bin/ls  
-rwxr-xr-x 1 root root 118280 Sep  2 2014 /bin/ls
```

## La home directory

- a ciascun utente viene assegnata una directory nella quale è libero di effettuare tutte le operazioni di creazione, spostamento, modifica, cancellazione su directory e file
- tale directory viene generalmente creata all'interno della directory

/home

- essa viene detta *home directory*
- l'associazione di un utente alla propria home directory è possibile grazie al processo di autenticazione

## La home directory

per esempio la home directory dell'utente `user1` può essere la seguente:

```
/home/user1
```

- dal momento che la home directory è una directory speciale che viene usata come punto di riferimento per ciascun utente, il suo percorso viene anche abbreviato utilizzando il carattere `~`
- per ciascun utente, la directory `~` corrisponde alla propria home directory
- il carattere `~` può essere usato come sostitutivo del percorso assoluto di una directory

esempi di percorsi validi sono i seguenti:

```
~/esercizi/soluzioni
```

```
~/programmi
```

```
~/
```

e vengono interpretate, per l'utente `user1`, rispettivamente come

```
/home/user1/esercizi/soluzioni
```

```
/home/user1/programmi
```

```
/home/user1
```

## Le directory `.` e `..`

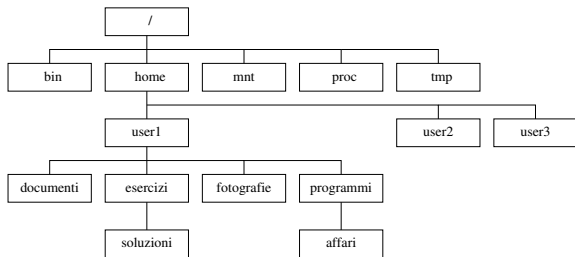
i nomi `.` e `..` sono nomi speciali di directory:

- `.` rappresenta la directory corrente
- `..` rappresenta la directory genitore di quella corrente

questi nomi di directory speciali possono essere utilizzati per formare dei percorsi sia relativi che assoluti



## Le directory . e ..



per esempio, il seguente percorso assoluto:

```
/home/./user1/esercizi/././soluzioni/./.../programmi/./fotografie
```

corrisponde alla directory identificata dal percorso assoluto

```
/home/user1/fotografie
```

## Il comando `cd`: spostamento tra directory

il comando `cd` cambia la directory corrente

per esempio

```
user2@europa:~$ pwd
/home/user2
user2@europa:~$ cd /home/user1/
user2@europa:~$ pwd
/home/user1
user2@europa:~$ cd programmi
user2@europa:~$ pwd
/home/user1/programmi
```

il comando `cd` è un comando interno alla shell

## Errore comune

- talvolta viene fatta confusione nello scrivere le giuste istruzioni per invocare il comando
- per esempio, il comando per spostarsi nella directory `programmi` è il seguente:

```
user1@europa:$ cd ./programmi
```

ed è ben diverso dal comando

```
user1@europa:$ ./programmi/cd
```

- il secondo comando richiede al sistema di **ESEGUIRE** il programma che si chiama `cd` e che si trova nella sottodirectory `programmi` della directory corrente!
- se tale comando non esiste, viene generato un messaggio di errore
- ma anche se esistesse, molto probabilmente non sarebbe il comando giusto per cambiare directory...

i comandi per creare e cancellare le directory sono i seguenti:

`mkdir dir` crea una directory di nome `d`

`rmdir dir` cancella la directory `d` purché essa sia vuota

## mkdir e rmdir: esempi di utilizzo

```
user1@europa:~$ mkdir tmp
user1@europa:~$ ls -l
total 20
drwxr-xr-x 3 user1 user1 4096 2008-07-23 14:14 documenti
drwxr-xr-x 3 user1 user1 4096 2008-07-23 14:12 esercizi
drwxr-xr-x 2 user1 user1 4096 2008-07-23 14:12 fotografie
drwxr-xr-x 2 user1 user1 4096 2008-07-23 14:12 programmi
drwxr-xr-x 2 user1 user1 4096 2008-07-23 14:32 tmp
user1@europa:~$ mkdir tmp/d1/d2
mkdir: cannot create directory `tmp/d1/d2': No such file or directory
user1@europa:~$ mkdir tmp/d1
user1@europa:~$ ls -l tmp/d1
total 0
user1@europa:~$ rmdir tmp
rmdir: failed to remove `tmp/': Directory not empty
user1@europa:~$ rmdir tmp/d1
user1@europa:~$ ls -l tmp/d1
ls: cannot access tmp/d1: No such file or directory
user1@europa:~$
```

```
mkdir tmp/d1/d2
```

- si è tentato di creare la directory `d2` come sottodirectory di `tmp/d1`
- la directory `tmp` esiste, ma la sua sottodirectory `d1` no
- è impossibile creare una directory `d2` contenuta in `d1` (quest'ultima non esiste!)

NOTA: per poter creare l'intero ramo di directory si può usare l'opzione `-p` nel seguente modo

```
mkdir -p tmp/d1/d2
```

```
rmdir tmp
```

- si è tentato di cancellare la directory `tmp`
- `tmp` è una directory non vuota
- non la si può quindi cancellare

## ls, mkdir e rmdir: errori

```
ls -l tmp/d1
```

- si è tentato di elencare il contenuto della directory `tmp/d1`
- la directory era stata precedentemente cancellata
- non esistendo più, è un errore tentare di listarne il contenuto



## Il comando `echo`: visualizzazione a terminale

- il comando `echo` stampa il testo che gli viene passato sulla riga di comando (ne fa l'eco)
- tale ripetizione porta alla visualizzazione del contenuto della riga di comando sullo schermo
- utile per stampare messaggi in uno script

### per esempio

```
user1@europa:~$ echo pranzo di lavoro  
pranzo di lavoro
```

visualizza sullo schermo il testo `pranzo di lavoro`

## La redirectione su file

- un modo veloce per inserire delle informazioni in un file è quello di utilizzare il comando `echo`
- per inserire nel file `agenda` i caratteri che formano la stringa `pranzo di lavoro` si può usare il comando

```
user1@europa:~$ echo pranzo di lavoro > agenda
```

il simbolo `>` è detto *operatore di redirectione*

la redirectione stampa su file ciò che normalmente viene visualizzato a video

- in altri termini, dirige l'output del comando precedente verso il file specificato dopo il simbolo `>`
- normalmente i comandi inviano il loro output verso ciò che si chiama *standard output*, o `stdout`
- lo standard output è un file associato solitamente al terminale (il video) quindi normalmente *scrivere sullo standard output* equivale a scrivere sul video

## La redirectione su file

```
user1@europa:~$ echo pranzo di lavoro > agenda
```

- il comando redirige lo standard output del comando `echo`, ovvero il testo che dovrebbe essere scritto sul terminale viene invece scritto sul file `agenda`
- se il file `agenda` non esiste, allora viene creato, mentre se il file esiste già, allora esso viene sovrascritto
- nel secondo caso, il contenuto del file preesistente viene perduto

l'operatore di redirectione va quindi utilizzato con cautela, per evitare di eliminare dei dati importanti, che sarebbe poi impossibile recuperare

## Il comando `cat`: visualizzazione del contenuto di file

per visualizzare sullo schermo il contenuto del file `agenda` si utilizza il comando `cat`:

```
user1@europa:~$ cat agenda  
pranzo di lavoro
```

il comando `cat` sta per *con(cat)enate*, in quanto concatena tutti i file specificati sulla sua linea di comando verso il suo standard output

## Il comando `cat`: scrittura su file

esempio:

```
user1@europa:~$ cat file1 file2 file3
```

concatena il contenuto dei tre file `file1`, `file2` e `file3`, verso il suo standard output, anch'esso generalmente associato al terminale

è possibile aggiungere (appendere) delle informazioni alla fine di un file esistente

esempio:

```
user1@europa:~$ echo cena fuori >> agenda
```

si utilizza l'operatore di redirezione formato dal “doppio maggiore” (>>) invece del comando di redirezione usuale

ora il file `agenda` avrà il seguente contenuto:

```
user1@europa:~$ cat agenda  
pranzo di lavoro  
cena fuori
```

- il file `agenda` esistente non è stato sovrascritto come nel caso in cui si fosse usato l'operatore `>`
- il nuovo testo viene *accodato* al file esistente



- la redirectione dell'output è una tecnica generale
- l'output di qualsiasi comando può essere rediretto

**esempio:** lo standard output del comando `cat` viene rediretto dal terminale verso file:

```
user1@europa:~$ cat agenda > copia_agenda
```

## cat e redirectione su dell'output

```
user1@europa:~$ cat agenda > copia_agenda
```

- reindirige l'output del comando `cat`, cioè il contenuto del file `agenda`, verso il file `copia_agenda`
- in pratica, in questo modo è stata effettuata la copia del file `agenda`

la redirectione su file dell'output di un comando mediante `>` causa la sovrascrittura del file eventualmente già presente avente lo stesso nome; i dati presenti nel file preesistente vanno così perduti

nota importante per quanto riguarda la visualizzazione a terminale dei messaggi di errore

- i messaggi di errore che raggiungono il terminale non vengono emessi sullo standard output ma su un altro canale, detto *standard error* (`stderr`)
- come lo standard output, lo standard error è solitamente associato al terminale
- in questo modo, quando l'output di un comando viene rediretto su file, eventuali messaggi di errore raggiungono comunque l'utente, venendo visualizzati sullo standard error che rimane associato al terminale

## Il comando `cp`: la copia di file

per copia di un file si intende la creazione di un secondo file che contiene esattamente le stesse informazioni del file di partenza;  
anche le directory possono essere copiate

```
cp f1 [f2 ...] dir
```

- viene creata la copia dei file `f1 [ ... ]` dentro la directory `dir`
- `dir` deve essere una directory esistente
- se `f1` esiste già dentro `dir`, il file viene sovrascritto

## Il comando `cp`: la copia di file

```
cp f1 f2
```

crea una copia del file `f1` di nome `f2` nella directory corrente

- `f2` deve essere un file (altrimenti vedi caso precedente)
- se `f2` esiste già viene sovrascritto

## Il comando `cp`: esempio

```
user1@europa:~$ cp agenda contatti ../tmp
user1@europa:~$ ls -l ../tmp/
total 8
-rw-r--r-- 1 user1 user1 28 2008-07-23 14:51 agenda
-rw-r--r-- 1 user1 user1 19 2008-07-23 14:51 contatti
user1@europa:~$ echo domani partita a tennis >> agenda
user1@europa:~$ cp agenda ../tmp/
user1@europa:~$ ls -l ../tmp/
total 8
-rw-r--r-- 1 user1 user1 52 2008-07-23 14:52 agenda
-rw-r--r-- 1 user1 user1 19 2008-07-23 14:51 contatti
user1@europa:~$
```

- si noti che listando la seconda volta il contenuto della directory `tmp` si vede come il file `agenda` sia stato sovrascritto
- ha infatti una dimensione maggiore (52 byte invece che 28), poiché contiene anche l'ultima stringa inserita

## La copia ricorsiva

```
cp -r f1 [f2 ...] dir
```

se `f1 [...]` è una `directory`, essa viene copiata ricorsivamente, cioè insieme alle sue `subdirectory` e tutti i file contenuti

## La copia ricorsiva: esempio

```
user1@europa:~$ ls documenti/  
affari/  agenda  contatti  
user1@europa:~$ cp -r documenti tmp/  
user1@europa:~$ ls -RC tmp/  
tmp/:  
documenti  
  
tmp/documenti:  
affari  agenda  contatti  
  
tmp/documenti/affari:  
user1@europa:~$
```



## Il comando `mv`: lo spostamento di file

```
mv f1 [f2 ...] dir
```

- sposta gli oggetti `f1...`, siano essi file o directory, dentro la directory `dir`
- `dir` deve esistere come directory
- se `f1` esiste già dentro `dir` il file viene sovrascritto
- `f1 [f2 ...]` può essere una directory, la quale verrà copiata ricorsivamente dentro `dir`

## Il comando `mv`: la rinominazione di file

```
mv f1 f2
```

- cambia il nome di `f1` in `f2`
- `f2` deve essere un file o non esistere (altrimenti vedi caso precedente)
- se `f2` esiste già viene sovrascritto

## Il comando mv: esempio

```
user1@europa:~$ ls tmp
user1@europa:~$ echo ciao > tmp/saluto
user1@europa:~$ ls tmp
saluto
user1@europa:~$ mv tmp/saluto .
user1@europa:~$ ls tmp
user1@europa:~$ ls
documenti  esercizi  fotografie  programmi  saluto  tmp
user1@europa:~$ ls tmp/saluto
ls: cannot access tmp/saluto: No such file or directory
user1@europa:~$ mv saluto tmp/ciao.file
user1@europa:~$ ls
documenti  esercizi  fotografie  programmi  tmp
user1@europa:~$ ls tmp
ciao.file
user1@europa:~$ cat tmp/ciao.file
ciao
user1@europa:~$
```

## Il comando `rm`: la cancellazione di file

```
rm f1 [f2 ...]
```

- cancella i file `f1 [f2 ...]`

```
user1@europa:~$ echo ciao > tmp/saluto
```

```
user1@europa:~$ ls tmp
```

```
saluto
```

```
user1@europa:~$ rm tmp/saluto
```

```
user1@europa:~$ ls tmp
```

```
user1@europa:~$
```

## Il comando `rm`: la cancellazione ricorsiva di file

```
rm -r dir
```

- cancella ricorsivamente la directory `dir` insieme a tutte le sotto-directory e tutti i file contenuti

## Il comando `rm`: la cancellazione ricorsiva di file

```
rm -fr dir
```

- l'argomento `-f` (“force”) permette di imporre la cancellazione senza una richiesta di conferma
- le opzioni `-f` e `-r` sono accorpate
- viene effettuata l'eliminazione della directory `dir` e tutto il suo contenuto senza chiedere alcuna conferma all'utente

## Il comando `rm`: la cancellazione ricorsiva di file

esempio:

```
user1@europa:~$ ls -r documenti/
```

```
contatti agenda affari
```

```
user1@europa:~$ ls -R documenti/
```

```
documenti/:
```

```
affari agenda contatti
```

```
documenti/affari:
```

```
user1@europa:~$ rm -fr documenti/affari
```

```
user1@europa:~$ ls -R documenti/
```

```
documenti/:
```

```
agenda contatti
```

si tratta di speciali caratteri che servono ad individuare più file in una sola invocazione di un comando

in particolare, sono disponibili le seguenti wildcard:

- \* indica *tutti i caratteri*
- ? indica *uno e un solo carattere*



## Le wildcard: esempi d'uso

```
ls ag*
```

- elenca tutti i file che iniziano per “ag” e sono seguiti da una qualsiasi combinazione di caratteri
- esempi: agenda1, agenda2.txt, aggiornamento.dat

il carattere “punto” viene trattato come un qualsiasi altro carattere (l'estensione non viene gestita automaticamente)

## Le wildcard: esempi d'uso

```
rm ag*t
```

- cancella tutti i file che iniziano con “ag” e finiscono con “t”
- tra “ag” e “t” ci può essere un qualsiasi numero di caratteri (anche 0) qualsiasi
- esempi: vengono cancellati `agenda.txt`, `agt`, `ag_qualsiasi-testo.t`
- il file `agtx` non viene invece cancellato

## Le wildcard: esempi d'uso

```
rm -fr *
```

- cancella tutti i file e le directory presenti nella directory corrente

NOTA:

```
rm -fr documenti/*
```

```
rm -fr documenti/
```

- sono leggermente diversi tra loro
- il primo cancella ricorsivamente tutto *il contenuto* della directory `documenti`, ma **NON CANCELLA** la directory `documenti` stessa
- il secondo comando cancella la directory `documenti` con tutto il suo contenuto

## Le wildcard: esempi d'uso

```
ls img?
```

- elenca tutti i file che iniziano con i caratteri “img”, seguiti da uno e un solo carattere
- esempi: vengono elencati i file `img1`, `img2`
- non vengono elencati i file `img`, `img12`, `im_g`

altro esempio, più vicino alla comune esperienza di programmazione in C:

```
ls programma.?
```

- elenca i file `programma.c`, `programma.h` e `programma.o`
- essi costituiscono i tipici file collegati al processo di realizzazione (scrittura del sorgente e compilazione) di un programma in C

## Attenzione alla pericolosità di alcuni comandi

- i comandi `rm`, `cp`, `mv` possono cancellare dei file, o sovrascriverli cancellando l'eventuale contenuto preesistente
- permettono di specificare l'argomento `-i` ("interattivo"), nel qual caso viene chiesta conferma all'utente per ogni cancellazione effettuata

in generale non c'è modo di recuperare i file una volta che sono stati cancellati

- `rm *` e `rm -r dir` sono comandi molto pericolosi, a maggior ragione se vengono utilizzati congiuntamente
- su molte macchine l'amministratore di sistema decide che `rm` è equivalente a `rm -i`, nel qual caso viene sempre chiesta conferma quando si cancellano i file

i comandi disponibili in un sistema Unix non sono altro che dei programmi eseguibili fisicamente presenti sul disco, i quali vengono eseguiti quando vengono invocati da linea di comando

- in tal senso, il compito principale della shell è quello di eseguire i programmi relativi ai comandi desiderati.
- la maggior parte dei comandi più utilizzati risiede nelle directory di sistema

```
/bin
```

```
/usr/bin
```

visualizzando il contenuto di tali directory con

```
ls /bin
```

```
ls /usr/bin
```

si possono scoprire molti comandi non descritti in questo documento



## La variabile PATH

in genere per l'esecuzione di un programma è necessario specificare esattamente il percorso (relativo o assoluto) del programma stesso

per esempio, è possibile invocare il comando `rmdir` con:

```
/bin/rmdir directory-da-cancellare
```

- il fatto che i comandi illustrati finora non richiedano la specifica di tutto il percorso per essere invocati è dovuto al fatto che è possibile specificare alla shell alcune directory nella quale ricercare i comandi che vengono invocati senza specificare il percorso
- per maggiori informazioni è necessario documentarsi su concetti quali le variabili di ambiente, e in particolare la variabile `PATH`

si considerino i due seguenti comandi:

```
$ ../rm nomefile
```

```
$ rm ../nomefile
```

- il primo comando invoca il programma `rm` presente nella directory `..` della directory corrente passandogli come argomento la stringa `nomefile`
- il secondo comando invoca il programma `rm` presente nella directory di sistema e cancella (o tenta di cancellare, se il file non esiste...) il file di nome `../nomefile`

## Il comando `man`: visualizzazione del manuale

è possibile utilizzare il comando `man` per visualizzare il manuale relativo a ogni comando disponibile in un sistema Unix, ovvero la relativa documentazione

la sua sintassi è

```
man [sezione] comando
```

per esempio, i comandi

```
user1@europa:~$ man cp
```

```
user1@europa:~$ man ls
```

```
user1@europa:~$ man man
```

visualizzano rispettivamente il manuale del comando `cp`, di `ls` e di `man` stesso (anche `man` è un comando!)

## Il comando `man`: visualizzazione del manuale

il manuale dei comandi è diviso in sezioni numerate da 1 a 9:

- 1 programmi eseguibili e comandi di shell
- 2 chiamate di sistema, ovvero le funzioni fornite dal kernel
- 3 chiamate di libreria
- 4 file speciali, tipicamente presenti nella directory `/dev`
- 5 formati di file e convenzioni, ad esempio `/etc/passwd`
- 6 giochi
- 7 pagine varie
- 8 comandi di amministrazione normalmente utilizzabili soltanto da `root`
- 9 funzioni del kernel (non standard)

## Il manuale dei comandi builtin

- alcuni comandi sono interni all'interprete (*builtin* della shell), cioè non corrispondono a programmi fisicamente presenti sul disco
- la gestione di questi comandi è programmata direttamente all'interno della shell
- un esempio di comandi builtin è `cd`

il comando

```
user1@europa:~$ man cd  
No manual entry for cd
```

tipicamente non da risultato

per conoscere i dettagli dei comandi interni basta invocare il comando

```
man sh
```

- viene visualizzata la pagina di manuale del programma `sh`, il quale non è altro che la shell stessa
- su alcuni sistemi Linux la pagina di manuale dei comandi builtin viene visualizzata correttamente, poichè la pagina di manuale è stata “estratta” da quella della shell per rendere più immediata la consultazione