Tutorial sulla Shell Linux

Sistemi Operativi

C.d.L. in Informatica (laurea triennale)
Dipartimento di Matematica e Informatica – Catania

A.A. 2021-2022

Cos'é l'autenticazione?

A differenza del sistema MS-DOS e dei primi sistemi Windows, i sistemi UNIX sono stati da sempre multi-utente. La stessa macchina può essere utilizzata da più utenti contemporaneamente (localmente e/o in remoto).

Ad ogni utente è associato un username ed una password che permettono di identificarlo al momento del login. I dati di tutti gli utenti del sistema sono memorizzati all'interno del file `/etc/passwd`.

Tutto il sistema è basato sui diritti di accesso: ogni oggetto (file, periferica, processo) ha delle strutture di accesso che decidono quali operazioni un dato utente è abilitato a fare. Queste strutture prevedono la suddivisione in gruppi logici (con intersezioni) per una maggiore flessibilità.

Ogni utente ha un proprio ambiente di lavoro perfettamente isolato dagli altri. Un utente, in genere, non può interferire nel lavoro degli altri utenti.

Esiste un utente particolare, chiamato root, che ha i massimi privilegi sul sistema: può fare quello che vuole. In genere è l'account dell'amministratore di sistema.

Ambiente iniziale

Una volta effettuato il login correttamente ci si ritrova all'interno del proprio spazio utente.
A secondo di come è configurato il nostro sistema ci possiamo ritrovare in due diverse situazioni:

- se il login è stato effettuato in modalità testuale, allora ci ritroveremo direttamente difronte ad una shell (sempre testuale); alcuni sistemi (sopratutto quelli meno recenti) utilizzano l'interfaccia testuale per default;
- il login viene effettuato attraverso una interfaccia grafica (desktop manager)che ci permette di introdurre le nostre credenziali (username/password) e quello che ci viene presentato dopo è un terminale grafico con una interfaccia user-friendly tipo Gnome o KDE; in questo caso, bisogna aprire manualmente un terminale per permetterci di lavorare con la shell.

La shell

Quello che ci viene presentato è un prompt che ci indica che il sistema è pronto a ricevere comandi da noi.

Esempio di prompt dei comandi

```
utente@host:directory$ _
```

A questo punto possiamo inserire i comandi che vengono poi mandati in esecuzione premendo il tasto di invio (o "enter"). Finché il comando non viene inviato in esecuzione, abbiamo la possibilità di modificarne la sintassi correggendolo. Infatti questo viene mantenuto in un buffer di input fino all'invio.

Alcuni tasti utili

`Ctrl-C`: interrompe l'esecuzione di un comando;
`Ctrl-S`: blocca l'output che scorre sullo schermo;
`Ctrl-Q`: sblocca lo stesso output.

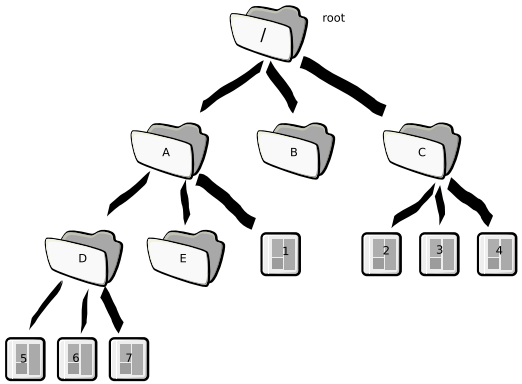
Il filesystem

Si tratta di una struttura (tipicamente ad albero) mantenuta su disco che contiene tutti i dati del S.O. e dei suoi utenti.

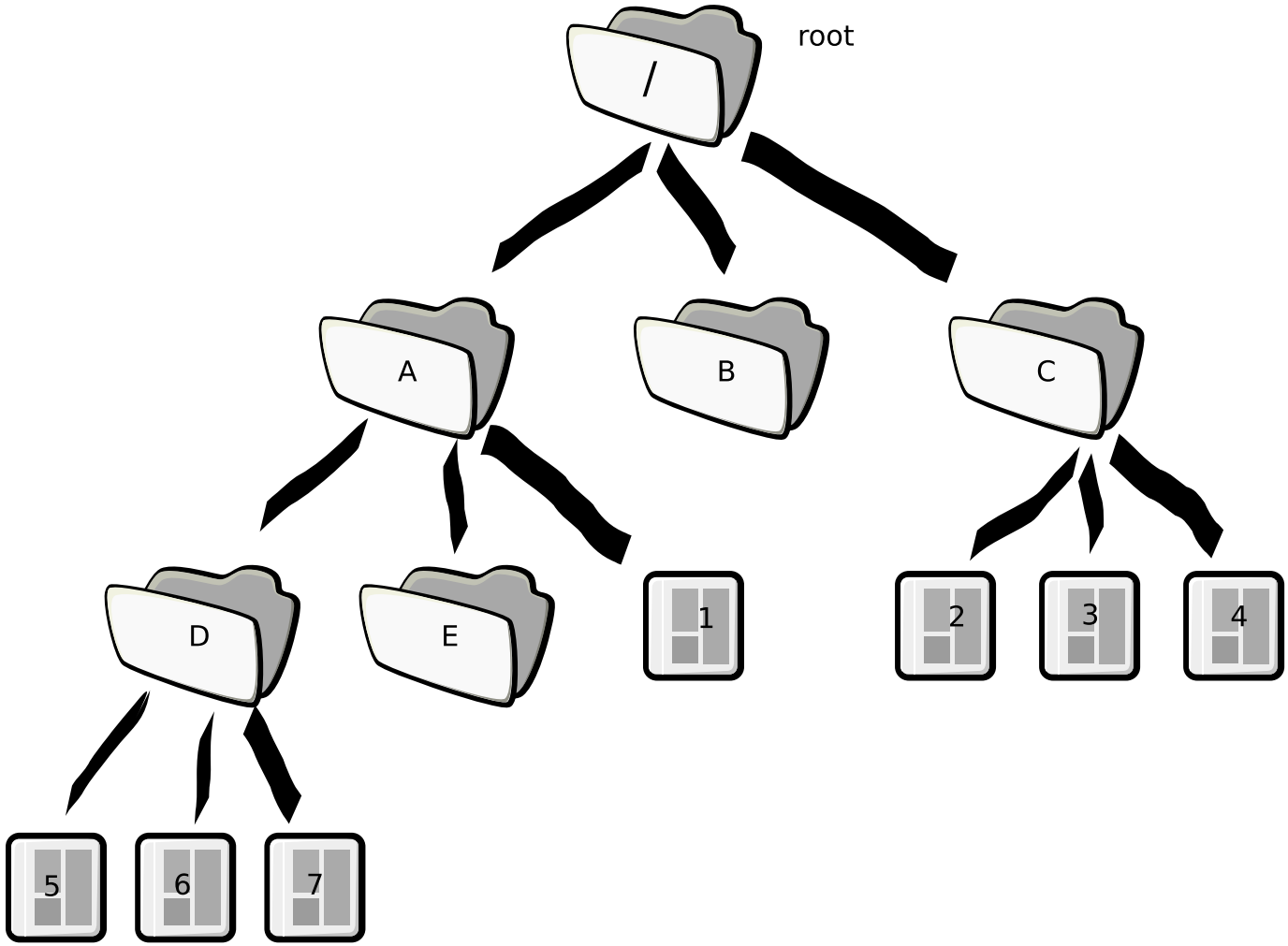
Il filesystem contiene i seguenti tipi di oggetti:

- file: unità di base di memorizzazione dei dati;
- directory: meccanismo di raggruppamento di altri oggetti in modo annidato, dando origine a strutture ad albero;
- altro.

Ogni filesystem ha un directory speciale che contiene tutti gli altri oggetti del filesystem (direttamente o indirettamente): root (o radice).



Esempio di filesystem



I path (1)

Per lavorare con gli oggetti del filesystem ci serve poterli identificare in un modo ben preciso. Vengono utilizzati due metodi principali:

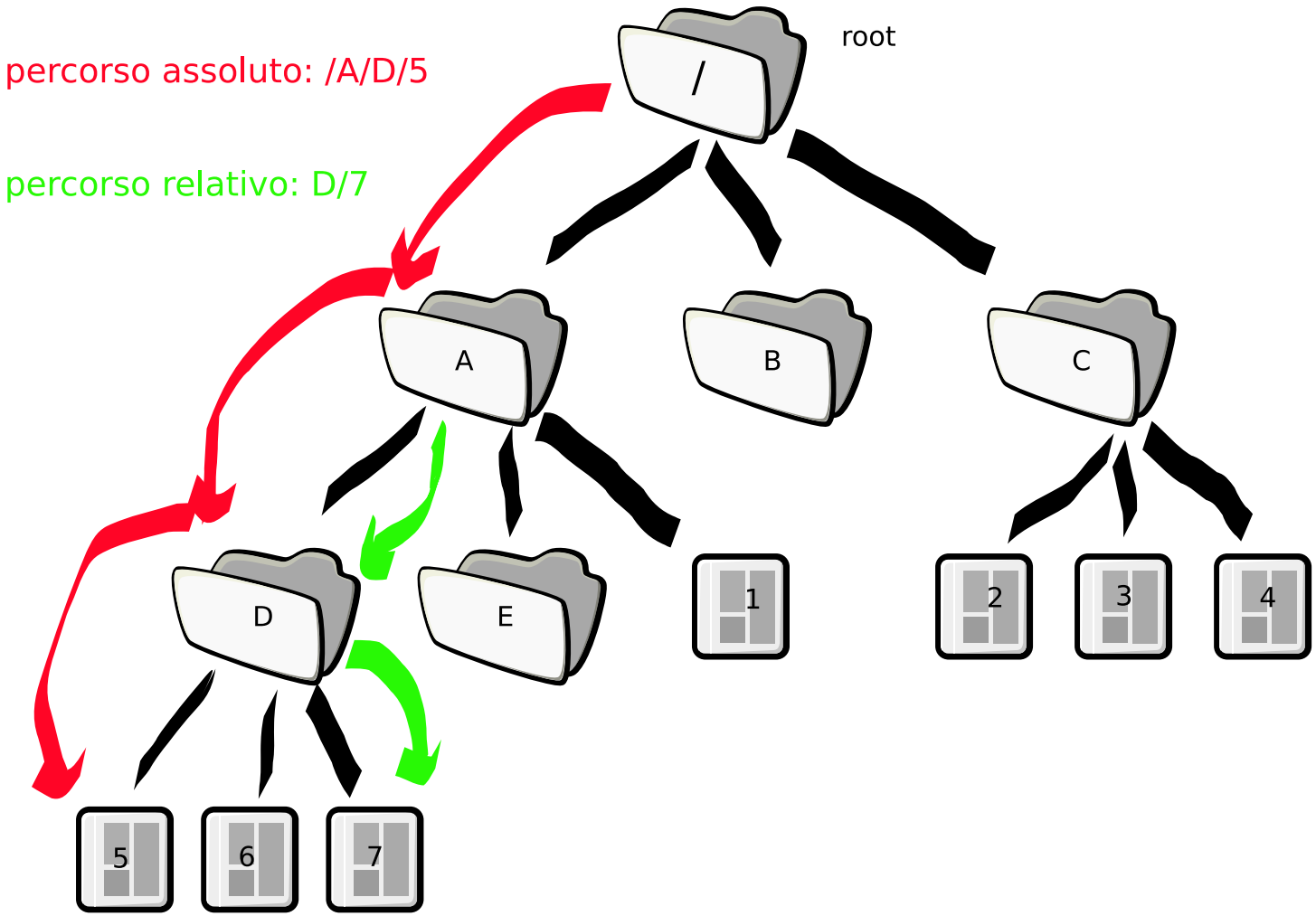
Percorso assoluto

Il percorso assoluto (o "absolute path") di un file è il percorso che va dalla radice del filesystem allo stesso.
Il percorso parte dalla root quindi inizia con il carattere `/` e ogni elemento del path è separato dal precedente da un altro carattere `/`.
Esempio: `/home/utente/file.txt`

Percorso relativo

Fissando logicamente una particolare directory (in genere quella corrente) è possibile identificare un file attraverso il percorso relativo (o "relative path") che va da tale directory ad esso.
Un percorso relativo non inizia mai con il carattere `/`.
Esempio: `utente/documenti/cv.pdf`

I path (2)


percorso assoluto: /A/D/5
percorso relativo: D/7
root
/
A
B
C
D
E
1
2
3
4
5
6
7

I path (3)

Osservazione

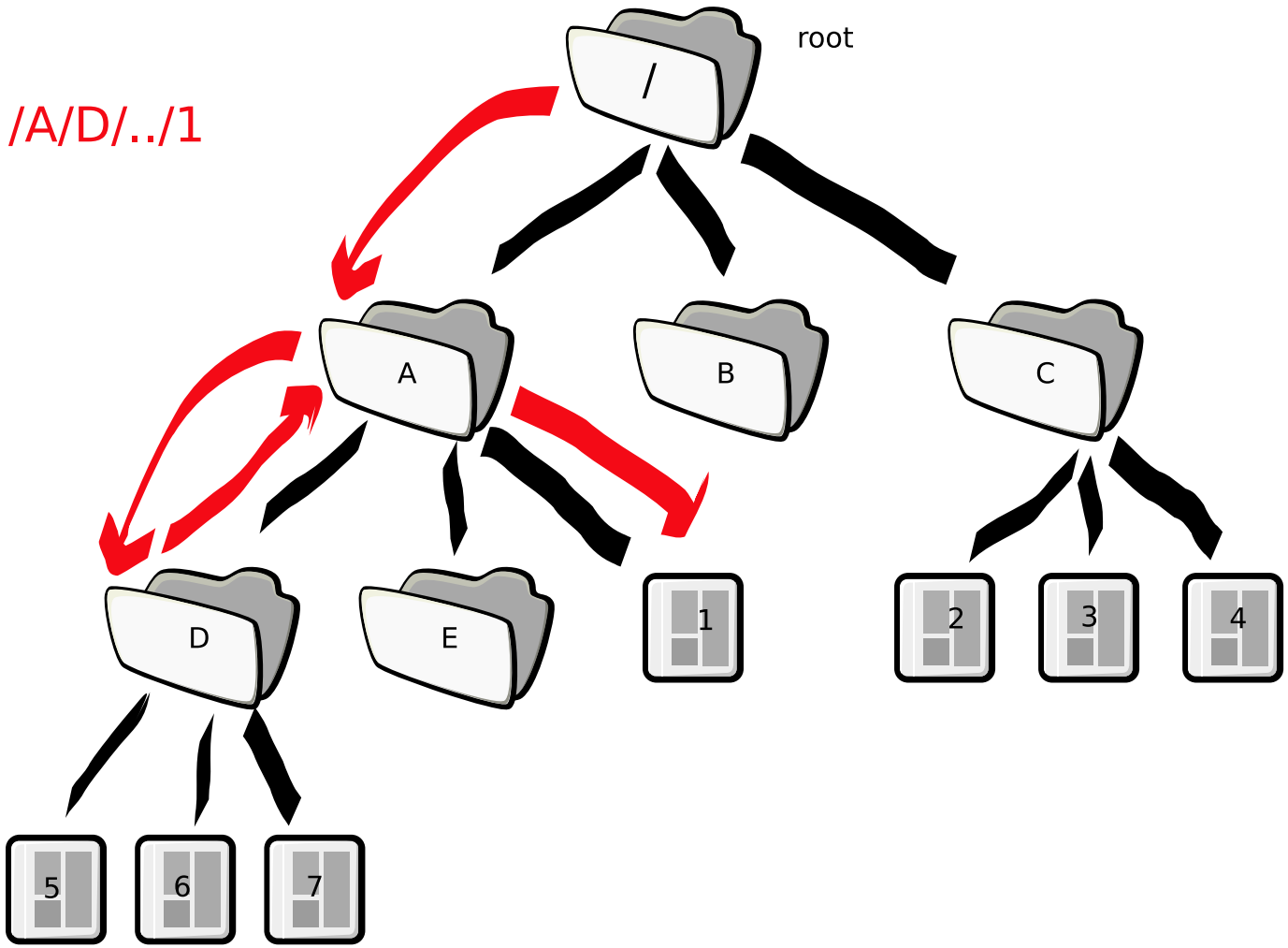
In effetti il percorso assoluto di un file è il suo percorso relativo rispetto alla root.

Nella costruzione dei percorsi possiamo utilizzare due "cartelle virtuali" che esistono in ogni directory:

- la cartella `.`: indica la cartella stessa ("auto-referenziamento");
- la cartella `..`: indica la cartella genitore.

La cartella speciale `..` risulterà molto utile per navigare all'interno del nostro filesystem.

I path (4)



File speciali

In ambiente UNIX esistono dei file speciali chiamati file di dispositivo: identificano una particolare periferica del sistema ed attraverso esso il S.O. e i programmi possono interagire con tale periferica.
In genere risiedono nella cartella predefinita `/dev/`.

Ne esistono di due tipi:

- a caratteri: le operazioni sul dispositivo vengono fatte per flussi di input/output con il carattere come unità di base. Esempio: tastiera, stampante, scheda audio;
- a blocchi: si opera con modalità ad accesso casuale e si agisce per blocchi. In genere riguarda i dischi fissi ed i CD/DVD.

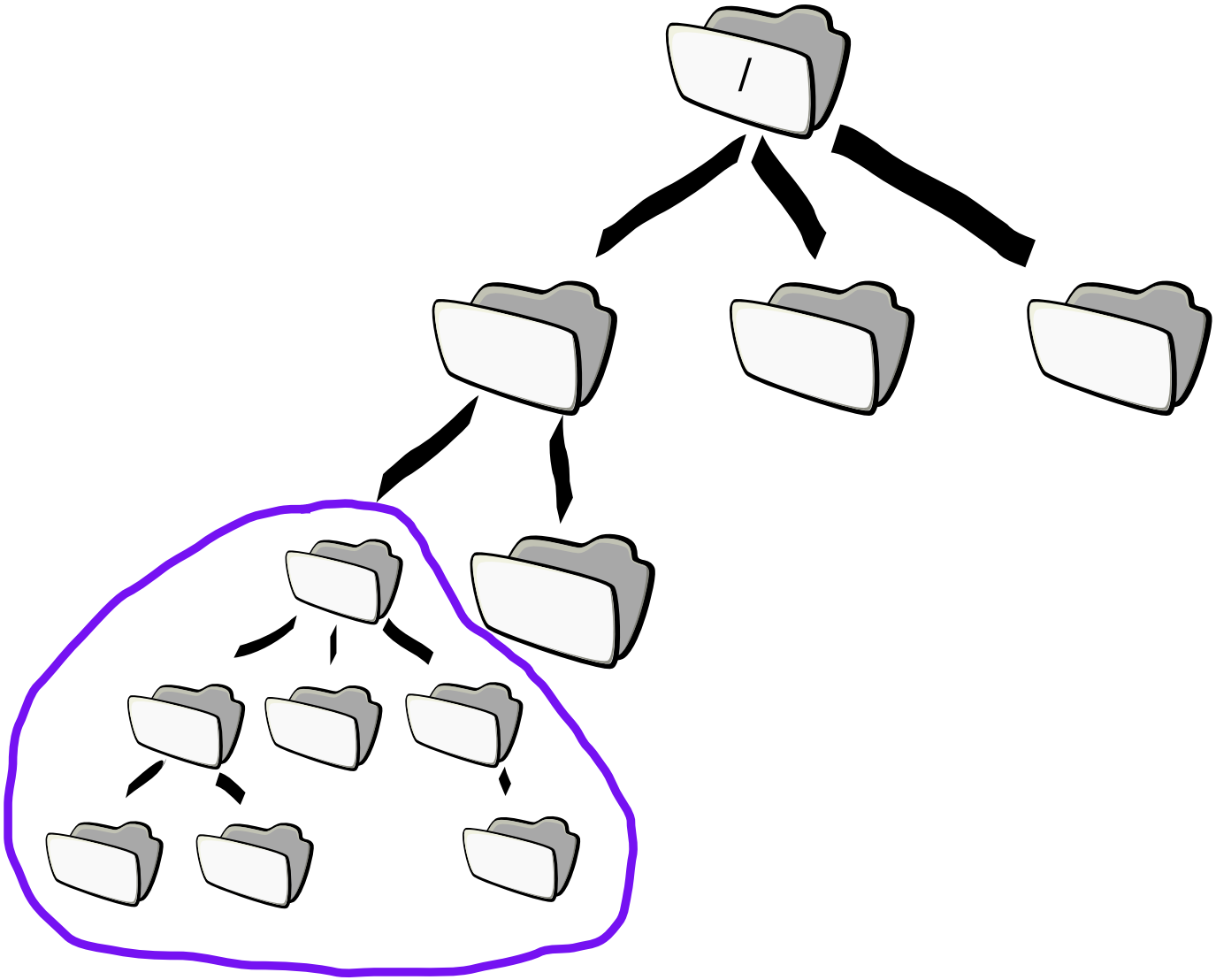
Esempi: dischi SATA (`/dev/sda`,`/dev/sdb`,...), partizioni all'interno dei dischi (`/dev/sda1`,`/dev/sda2`,...), terminali testuali (`/dev/tty1`), schede audio (`/dev/dsp`).

Più di un filesystem

In genere in un sistema esistono più di un filesystem: ne necessita uno per un eventuale CD-ROM inserito nel lettore, uno per ogni partizione sul disco, ecc.
Sui sistemi MS-DOS/Windows siamo abituati ad identificare ogni possibile filesystem con una diversa lettere dell'alfabeto: `A:`, `C:`, ...

Nei sistemi UNIX tutto questo viene evitato montando i filesystem in cascata. Esiste un filesystem principale (in genere quello da cui viene caricato il S.O.) e la sua radice sarà la root di tutto il sistema. Ogni filesystem secondario viene "montato" come se fosse un ramo dell'albero principale.
In questo modo si viene a creare un unico grande albero ed ogni file del sistema può essere identificato con un path assoluto che parta dalla root del sistema `/`.
In genere i filesystem secondari vengono montati della cartella standard `/mnt/` e quelli delle unità removibili (tipo floppy-disk o chiavette USB) in `/media/`.

Esempio di mount



Un tipico filesystem UNIX

Ecco alcune directory standard che si trovano su tutti i sistemi UNIX:

- `/home/`: ogni utente ha una propria home directory in cui i propri dati e le sue configurazioni dei programmi vengono mantenute (separatamente da quelli degli altri utenti); tutte le home degli utenti stanno in `/home/`;
- `/etc/`: vi risiedono tutti i file di configurazione dei programmi installati compresi gli script di avvio del sistema;
- `/bin/`: contiene i file eseguibili di buona parte delle utility del sistema;
- `/sbin/`: contiene le utility destinate all'amministrazione che solo l'amministratore può utilizzare;
- `/lib/`: contiene le librerie di sistema;
- `/proc/`: una directory virtuale che rappresenta alcuni aspetti interni del S.O., tipo processi, periferiche, ecc.
- `/usr/`: contiene gli eseguibili e i dati di tutto il resto dei programmi installati;
- `/var/`: contiene alcuni database, cache e dati temporanei.

I comandi

I comandi che si possono invocare da una shell sono di due tipi:

- comandi esterni: si tratta di applicazioni indipendenti dalla shell che sono memorizzate all'interno del filesystem ed eseguite indipendentemente dal processo della shell; possono andare dalla semplice utility testuale ad vere e proprie applicazioni grafiche complete;
- comandi builtin: vengono eseguiti dalla shell stessa senza coinvolgere processi aggiuntivi. Tipicamente questi comandi vengono usati per manipolare le impostazioni interne della shell e per aumentare l'efficienza dei comandi usati più frequentemente. Fanno parte dei comandi builtin, inoltre, anche tutti i costrutti del linguaggio di scripting della shell.

I parametri

Tipicamente un comando può richiedere dei parametri. Si inseriscono dopo il nome del comando separati da uno spazio. Questi parametri possono essere:

- file: il percorso (assoluto o relativo) al/ai file su cui il comando deve lavorare;
- dati: identificano dei dati da passare ed in genere sono passato sotto forma di stringa con una formattazione opportuna;
- switch o opzioni: sono dei parametri che, usualmente, iniziano con un carattere '`-`'. Danno al comando alcune indicazioni sull'esecuzione del suo compito. Spesso la stessa opzione ha più di una sintassi: una breve ed una lunga. Quella breve è in genere del tipo `-h` (dove `h` è un singolo carattere). Quella lunga inizia con un doppio trattino `--` ed ha in genere un nome più mnemonico, tipo `--help`.

Spesso le opzioni si possono collassare: al posto di due opzioni `-a -b`, si può spesso utilizzare `-ab`. Alcuni parametri sono opzionali e nella sintassi vengono indicati tra parentesi quadre `[-]`. Esempio: `cal [[mese] anno]`.

Cerchiamo un po' di aiuto

I comandi UNIX sono tanti e le opzioni sono centinaia. Generalmente ogni comando è capace di fornire una breve descrizione delle sue funzionalità e delle sue principali opzioni quando viene impiegata l'opzione `-h` o `--help`.

Per avere più aiuto ci sono altri metodi:

- `apropos`: Sintassi: `apropos stringa`
  Visualizza la lista di tutti i comandi che contengono la stringa passata per parametro nella loro descrizione sintetica;
- `whatis`: Sintassi: `whatis nome_comando`
  Visualizza la descrizione sintetica del comando passato come parametro;
- `man`: Sintassi: `man nome_comando`
  Visualizza la pagina del manuale in linea relativa al comando passato come parametro.

La lingua in cui le informazioni vengono fornite dipende dal sistema.

E qui cosa c'è?

`ls`

Sintassi (semplificata): `ls [-l] [-a] [-R] [pathname...]`

- `-l`: visualizza informazioni dettagliate
- `-a`: visualizza anche i file nascosti
- `-R`: visualizza anche il contenuto delle cartelle ricorsivamente
- `pathname`: oggetto del filesystem sul quale visualizzare le informazioni

Il comando `ls` consente di visualizzare informazioni sugli oggetti presenti nel filesystem. Mediante il parametro `pathname` è possibile identificare l'oggetto sul quale reperire le informazioni. Se `pathname` è una directory allora viene visualizzato tutto il contenuto di quest'ultima. `ls` lanciato senza parametri assume come pathname la directory corrente. E' possibile specificare più pathname sulla stessa linea di comando, il comando visualizzerà informazioni per ciascuno di essi.

Ci sono tante altre opzioni.

I dettagli sui file

Quando viene utilizzata l'opzione `-l` con `ls`, questo riporta una lista in cui per ogni riga ci sono i dettagli su un file o directory.

La tipica riga potrebbe essere la seguente:

```
-rwxr-xr-x 1 utente gruppo 5642 2005-09-20 10:12 script.sh
```

- i permessi di accesso dell'utente, gruppo e degli altri;
- il numero di hard link (lo vedremo dopo);
- il proprietario ed il gruppo;
- le dimensioni del file;
- la data e l'ora di creazione del file;
- il nome del file.

I permessi di accesso

```
-rwxr-xr-x 1 utente gruppo 5642 2005-09-20 10:12 script.sh
```

Si possono vedere come tre triple di permessi.
I permessi sono quelli di:

- `r`: lettura;
- `w`: scrittura;
- `x`: esecuzione.

Le triple riguardano: il proprietario del file, gli appartenenti al gruppo, tutti gli altri.

Il prima carattere è destinato a contenere dei flag speciali:

- `d`: si tratta di una directory;
- `l`: si tratta di un soft link (lo vedremo dopo).

Per le directory il simbolo `x` indica il permesso di attraversamento.

Esempio di uso di `ls`

```
$ ls
cartella  esempio2.txt  esempio.txt  script.sh

$ ls -la
drwxr-xr-x  3 mario mario 4096 2005-09-20 12:30 .
drwxr-xr-x  3 mario mario 4096 2005-09-20 12:28 ..
drwxr-xr-x  2 mario mario 4096 2005-09-20 12:30 cartella
-rw-r--r--  1 mario mario   27 2005-09-20 12:30 esempio2.txt
-rw-r--r--  1 mario mario   21 2005-09-20 12:29 esempio.txt
-rwxr-xr-x  1 mario mario   10 2005-09-20 12:30 script.sh

$ ls -R
.:
cartella  esempio2.txt  esempio.txt  script.sh

./cartella:
agenda.txt  eseguimi.sh

$ ls -l cartella/
-rw-r--r--  1 mario mario 21 2005-09-20 12:33 agenda.txt
-rwxr-xr-x  1 mario mario 10 2005-09-20 12:33 eseguimi.sh
```

I metacaratteri

Per abbreviare il nome di un file da specificare o per specificarne più di uno si possono utilizzare i metacaratteri:

- *: rappresenta una qualunque stringa di 0 o più caratteri;
- ?: rappresenta un qualunque carattere;
- [ ]: singolo carattere tra quelli elencati;
- { }: stringa tra quelle elencate.

```
$ ls
esempio2.txt  esempio3.txt  esempio.txt  script.sh scheda.pdf documento.pdf prova.sh

$ ls esempio*.txt
esempio2.txt  esempio3.txt  esempio.txt

$ ls esempio?.txt
esempio2.txt  esempio3.txt

$ ls *.{txt,pdf}
documento.pdf  esempio2.txt  esempio3.txt  esempio.txt  scheda.pdf

$ ls /dev/tty[acd][2-5]
/dev/ttya2  /dev/ttya4  /dev/ttyc2  /dev/ttyc4  /dev/ttyd2  /dev/ttyd4
/dev/ttya3  /dev/ttya5  /dev/ttyc3  /dev/ttyc5  /dev/ttyd3  /dev/ttyd5
```

Cambiamo directory

cd

Sintassi : `cd [pathname]`

- `pathname`: nuova directory corrente

Il comando `cd` consente di navigare all'interno del filesystem cambiando di volta in volta la directory corrente. Se si omette il parametro pathname la directory corrente viene impostata alla home directory per l'utente attuale (in genere `/home/nomeutente`).

Si possono usare sia i pathname assoluti che relativi, compresa l'utilissima directory virtuale `...`.

Dove mi trovo? Creiamo una directory

pwd

Sintassi : `pwd`

Il comando `pwd` (*present working directory*) permette di conoscere il pathname assoluto della directory corrente.

mkdir

Sintassi : `mkdir [-p] pathname...`

- `-p`: non genera errori se il pathname esiste già ed inoltre crea tutte le directory necessarie per creare il pathname passato
- `pathname`: pathname da creare

Il comando `mkdir` modifica la struttura del filesystem creando le directory specificate mediante i parametri. E' possibile specificare più pathname sulla stessa linea di comando, il comando verrà eseguito per ciascuno di essi.

Cancelliamo questa directory!

rmdir

Sintassi : `rmdir [-p] pathname...`

- `-p`: tenta di rimuovere tutte le directory che compongono il pathname; un comando `rmdir -p /a/b/c/` è equivalente a `rmdir /a/b/c /a/b/ /a/`
- `pathname`: directory da eliminare

Il comando `rmdir` modifica la struttura del filesystem cancellando le directory specificate mediante i parametri. E' possibile specificare più pathname sulla stessa linea di comando, il comando verrà eseguito per ciascuno di essi.
Le directory devono essere vuote altrimenti non vengono cancellate.

Copiamo questi file

cp

Sintassi (semplificata): `cp [-R] [-i] source... dest`

- `-R`: copia tutto il contenuto di `source` se è una directory
- `-i`: chiede conferma prima di sovrascrivere i file
- `source`: oggetti da copiare in `dest`; si possono specificare più sorgenti nella stessa riga di comando
- `dest`: destinazione in cui copiare i file specificati

Il comando `cp` copia i file specificati con `source...` in `dest`. Se si specifica una sola sorgente (ad esempio un file) allora specificando il pathname completo di un file come destinazione, questo viene copiato e rinominato. Se si specifica una cartella esistente come destinazione, i file vengono copiati al suo interno. Se si specificano più sorgenti, allora la destinazione deve essere necessariamente una cartella (esistente).

Cancelliamo questi file

rm

Sintassi: `rm [-r] [-i] [-f] pathname...`

- `-r`: se `pathname` è una directory, elimina ricorsivamente tutti i file o cartelle contenuti al suo interno
- `-i`: chiede conferma prima di cancellare ogni file
- `-f`: cancella gli oggetti senza chiedere conferma
- `pathname`: oggetti da eliminare

Il comando `rm` elimina tutti i file specificati con i parametri `pathname...`. Se viene specificato il parametro `-r` vengono eliminate ricorsivamente tutte le directory presenti nel sottoalbero della directory specificata con pathname.

Usando l'opzione `-r` ed abbastanza permessi è possibile danneggiare irrimediabilmente il sistema. Da non eseguire mai (da root): `rm -rf /`

Spostiamo quei file

mv

Sintassi: `mv source... dest`

- `source`: file o directory da spostare
- `dest`: file o directory di destinazione

Il comando `mv` sposta il file o directory sorgente nella destinazione specificata. Se si specifica solo una sorgente (un file o una directory) e la destinazione non esiste, allora la sorgente viene rinominata oltre che spostata. Se si specificano più parametri `source`, `dest` deve essere una directory.
Nota: spostare una directory in un'altra implica lo spostamento di tutto il sottoalbero della directory sorgente.

Redirezione dell'I/O

Ogni processo ha 3 flussi di dati:

- standard input: da cui prende il suo input; in genere corrisponde alla tastiera;
- standard output: in cui invia il suo output; in genere corrisponde al terminale video;
- standard error: in cui emette gli eventuali messaggi di error; anche qui si usa in genere il terminale.

Attraverso l'invocazione da riga di comando è possibile redirezionare tali sflussi su altri file.

- >: redireziona l'output su un file;
- >>: redireziona l'output su un file in modalità append;
- <: prende l'input da un file;
- 2>: redireziona lo standard error su un file.

Cosa c'è dentro quel file?

cat

Sintassi: `cat [pathname...]`

- `pathname`: file da visualizzare

Il comando `cat` permette di visualizzare (nel senso di mandare allo standard output) il contenuto di uno o più file.
Volendo si può invocare senza alcun parametro ed in questo caso `cat` prenderà il suo standard input come input (come la maggior parte dei comandi UNIX).

Nonostante l'apparenza questo comando risulta molto utile quando combinato con i meccanismi di comunicazione tra processi.

Dica "trentatre"

echo

Sintassi: `echo [-n] [-e] [stringa...]`

- `-n`: non manda a capo il carrello quando ha finito
- `-e`: permette l'uso di alcuni caratteri speciali
- `stringa`: stringa da visualizzare

Il comando `echo` da in output una stringa passata come parametro. Esempi di caratteri speciali che è possibile utilizzare utilizzando l'opzione `-e` sono: `\a` bell (campanello), `\n` new line, `\t` tabulazione, `\\` backslash, `\nnn` il carattere il cui codice ASCII (in ottale) è `nnn`.

Anche questo comando, combinato con i meccanismi di comunicazione tra i processi, può risultare molto utile.

Redirezione dell'I/O: esempi

```
$ ls > output.txt

$ cat output.txt
esempio2.txt  esempio3.txt  esempio.txt

$ echo Ciao >> output.txt
esempio2.txt  esempio3.txt

$ cat output.txt
esempio2.txt  esempio3.txt  esempio.txt
Ciao

$ cat < output.txt
esempio2.txt  esempio3.txt  esempio.txt
Ciao

$ man comandochenonesiste
Non cè' il manuale per comandochenonesiste

$ man comandochenonesiste 2> /dev/null
```

Versione più evolute del comando cat

more

Sintassi: `more [pathname...]`

- `pathname`: file da visualizzare

Il comando `more` si comporta come `cat` (inviando il suo standard input o il file specificato al suo output) con l'eccezione che se l'output è più lungo di una videata di schermo, ad ogni pagina viene fatta una pausa.

less

Sintassi: `less [pathname...]`

- `pathname`: file da visualizzare

Il comando `less` è una versione più evoluta di `more`: non solo permette di fare pause ad ogni pagina, ma permette anche di andare su e giù nell'output. Inoltre è possibile effettuare semplici ricerche interattive all'interno dell'input.

Le pipeline

Esiste anche un altro metodo di comunicazione tra i processi: le pipeline.
Due comandi possono essere messi in cascata collegando l'output del primo con l'input del secondo.
La sintassi è la seguente: `comando1 | comando2`
In realtà i due comandi vengono mandati in esecuzione contemporaneamente: il secondo aspetta che arrivi man mano l'output del primo.

Si possono mettere in comunicazione anche più di due comandi: `comando1 | comando2 | ... | comandon`. L'output della pipeline corrisponderà all'output dell'ultimo comando.

```
$ ls /usr/bin | more

$ ls | lpr
```

Impariamo a contare

wc

Sintassi: `wc [-c] [-w] [-l][pathname...]`

- `-c`, `-w`, `-l`: conteggia, rispettivamente, i caratteri/le parole (separate da spazi)/le righe
- `pathname`: file da analizzare

Il comando `wc` effettua l'analisi del suo standard input (se non vengono passati parametri) o dei file passati conteggiando il numero di byte, parole e/o righe. Se non si specifica una opzione particolare vengono riportati tutti e tre i conteggi. Si possono passare più file.

```
$ echo "Ciao a tutti." | wc -c -w
       3       14
$ wc /etc/passwd /etc/fstab /etc/group
  30   41 1317 /etc/passwd
  13   70  655 /etc/fstab
  55   55  728 /etc/group
  98  166 2700 totale
```

Mettiamo un po' d'ordine

`sort`

Sintassi (semplificata): `sort [-n] [-r] [-o file] [-t s] [-k s1[,s2]] [pathname]`

- `-n`: considera numerica (invece che testuale) la chiave di ordinamento
- `-r`: ordina in modo decrescente
- `-o file`: invia l'output su `file` anziché sullo standard output
- `-t s`: usa `s` come separatore di campo
- `-k s1,s2`: usa i campi da posizione `s1` a `s2` per l'ordinamento
- `pathname`: file da ordinare

Il comando `sort` ordina trattando ogni linea del suo input come una collezione di campi separati da delimitatori (default: spazi, tab, ecc.). L'ordinamento di default avviene in base al primo campo ed è alfabetico.

Esempi di utilizzo di sort

```
$ ls -1 | sort -r
script.sh
scheda.pdf
prova.sh
esempio3.txt
esempio2.txt
documento.pdf
conta.sh

$ cat /etc/passwd | sort -t: -k 3,3
root:x:0:0:root:/root:/bin/bash
mario:x:1000:1000:Mario,,,:/home/mario:/bin/bash
test:x:1001:1001:Test User,,,:/home/test:/bin/bash
...........
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh

$ cat /etc/passwd | sort -t: -k 3,3 -n
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
...........
test:x:1001:1001:Test User,,,:/home/test:/bin/bash
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh

$ ls /bin /sbin/ /usr/bin/ /usr/sbin/ | sort -o lista_comandi.txt
```

Testa e coda

head

Sintassi (semplificata): `head [-c q] [-n q] [pathname...]`

- `-c q`: mostra solo i primi `q` byte dell'input
- `-n q`: mostra solo le prime `q` righe dell'input
- `pathname`: file da analizzare

Il comando `head` mostra di default le 10 righe del suo input. Può accettare anche più file.

tail

Sintassi (semplificata): `tail [-c q] [-n q] [pathname...]` Il comando `tail` si comporta come `head` ma a partire dalla fine dell'input.

Esempio: `cat /etc/passwd | tail -n 2 | head -n 1`

Cambio dei permessi di accesso (1)

chmod

Sintassi: `chmod [-R] mode [pathname...]`

- `-R`: applica i permessi in modo ricorsivo alle sotto-cartelle
- `mode`: nuova maschera dei permessi
- `pathname`: oggetti a cui applicare la nuova maschera dei permessi

Il comando `chmod` permette di cambiare i permessi a file o cartelle.

parametro mode: sintassi numerica

Si usa un numero ottale a 3 cifre in cui ogni cifra corrisponde rispettivamente a: proprietario, gruppo e altri.
I diritti sono di lettura, scrittura ed esecuzione/attraversamento. Ognuno è rappresentato da un bit: 1 per abilitato, 0 per disabilitato.
Esempio:

`rw-r-----` $\longrightarrow$ `110 100 000` $\rightarrow$ `640`

`rwxr-xr-x` $\longrightarrow$ `111 101 101` $\rightarrow$ `755`

Cambio dei permessi di accesso (2)

parametro `mode`: sintassi simbolica

Si utilizza una stringa `mode` del tipo: `target grant permission`

- `target`: definisce a chi cambiare i permessi tra user (`u`), group (`g`), other (`o`) o all (`a`);
- `grant`: un carattere tra `+`, `-` e `=`. I caratteri `+` e `-` aggiungono e sottraggono i permessi specificati lasciando inalterati tutti gli altri. Il carattere `=` imposta la maschera esattamente ai permessi specificati;
- `permission`: è una sottostringa di `rwx` ed indica i permessi che si vogliono alterare.

Specifiche multiple vanno separate da virgola. Se `target` viene omesso, si sottintende `all`.

`rw-r-----` $\longleftrightarrow$ `u=rw,g=r,o-rwx`

`rw-rw-rw-` + `u+x,o-rw` $\longrightarrow$ `rwxrw----`

`rw-r--r--` + `a-r,u+r` $\longrightarrow$ `rw-------`

`rw-r--r--` + `+x` $\longrightarrow$ `rwxr-xr-x`

Questo è mio, quell'altro è tuo

chown

Sintassi: `chown [-R] owner[:group] [pathname...]`

- `-R`: esegue ricorsivamente le modifiche di proprietà
- `owner`: il nuovo proprietario
- `group`: il nuovo gruppo proprietario
- `pathname`: oggetti a cui cambiare la proprietà

Il comando `chown` permette di cambiare i proprietari associati ad uno o più file.

chgrp

Sintassi: `chgrp [-R] group [pathname...]`

- `-R`: esegue ricorsivamente le modifiche di proprietà
- `group`: il nuovo gruppo proprietario
- `pathname`: oggetti a cui cambiare la proprietà

Modalità di utilizzo degli utenti

Su un sistema UNIX è bene creare uno o più utenti "normali" per le operazioni quotidiane che non richiedono poteri speciali. In questo modo si limitano i rischi di danneggiare il sistema incautamente e, soprattutto, i danni che si possono causare se l'account viene compromesso da un entità estranea.
Si può usare il comando `adduser` per aggiungerne di nuovi ed il comando `passwd` per cambiare la propria password in qualunque momento.

L'utente amministratore root dovrebbe essere usato solo quando strettamente necessario per effettuare operazioni di manutenzione, aggiornamento o configurazione del sistema.
Invece di fare il login come root è possibile diventarlo temporaneamente usando il comando `su` (super user).
Si può diventare un qualunque utente (previa autenticazione) utilizzando `su nome_utente` o si può semplicemente eseguire un solo comando come se fossimo un dato utente con la sintassi `su -c "comando argomenti" nome_utente`.

Modalità di utilizzo degli utenti

Su alcuni sistemi Linux (Ubuntu e derivate) e su Mac OS X, l'utente speciale root non viene direttamente utilizzato. Esiste un gruppo di utenti amministratori: ognuno può ottenere i pieni diritti di amministrazione all'occorrenza.
Uno di questi utenti può eseguire con diritti superiori un qualunque comando preponendo il suffisso `sudo`. La sintassi completa diventa quindi `sudo comando argomenti`.
A verifica della propria identità, viene richiesta conferma della password dell'utente stesso (non di root, che potrebbe anche non essere impostata). Tale verifica verrà riproposta solo dopo un certo lasso di tempo.

Facciamo un po' di spazio

gzip

Sintassi (semplificata): `gzip [-d] [-c] [pathname...]`

- `-d`: decomprime invece che comprimere
- `-c`: il file compresso viene mandato allo standard output
- `pathname`: file da comprimere

Il comando `gzip` comprime, con appositi algoritmi, il contenuto dei file specificati o del suo standard input per occupare meno spazio. Se vengono specificati sulla riga di comando, i file vengono sostituiti da una versione compressa con estensione `.gz`. Se non viene passato nulla, lo standard input viene compresso e passato allo standard output (funziona da filtro).

Per decomprimere si può usare l'opzione `-d` oppure il comando `gunzip`. Per vedere il contenuto di un file compresso si può anche utilizzare il comando `zcat` al posto di `cat`. Esistono anche le varianti `zmore`, `zless`.

Voglio più spazio!

Esiste un'altra coppia di comandi: `bzip2` e `bunzip2`.
A differenza di `gzip`, questi utilizzano degli algoritmi di compressione più evoluti che permettono, in genere, di ottenere compressioni migliori (a scapito di una maggiore elaborazione e richiesta di memoria). La sintassi e le modalità di uso sono simili a quelle di `gzip` e `gunzip`.
Cambia l'estensione dei file compressi: `.bz2`

Esistono comandi specifici tipo: `bzcat`, `bzmore`, `bzless`.

Di recente si sta impiegando un ulteriore formato: `.xz`; per esso esistono i relativi comandi con analoga sintassi: `xz`, `unxz`, `uxcat`, `xzmore`, `xzless`, ...

Esempi di utilizzo dei comandi di compressione

```
$ ls -l config.*
-rw-r--r--  1 mario mario 2389 2005-09-23 12:41 config.txt
$ gzip config.txt
$ ls -l config.*
-rw-r--r--  1 mario mario 813 2005-09-23 12:41 config.txt.gz
$ gunzip config.txt.gz
$ ls -l config.*
-rw-r--r--  1 mario mario 2389 2005-09-23 12:41 config.txt

$ cat config.txt | bzip2 -c > config.txt.bz2
$ ls -l config.*
-rw-r--r--  1 mario mario 2389 2005-09-23 12:41 config.txt
-rw-r--r--  1 mario mario  722 2005-09-23 12:44 config.txt.bz2

$ bzcat config.txt.bz2 > config.txt

$ ls -R /etc/ | gzip | bzip2 | bzcat | zmore

$ cat /dev/cdrom | bzip2 > mio_cdrom.iso.bz2
```

Archiviare più file

I comandi UNIX tipo `gzip` hanno scopi ben diversi dai più noti programmi di compressione ZIP e RAR. Quest'ultimi hanno lo scopo di mettere in un archivio compresso più file se non interi alberi del filesystem.
I comandi tipo `gzip` sono comandi di flusso che invece hanno lo scopo di comprimere un flusso di dati che nel caso base può essere semplicemente un file.

Nota

Nel mondo UNIX esistono comandi che permettono di gestire i formati ZIP e RAR. Quest'ultimi però vanno considerati solo standard di fatto essendo formati proprietari (anche se aperti) creati da aziende.
Lo standard di compressione nei sistemi UNIX fanno riferimento a `gzip`, `bzip2` e, recentemente, `xz`.

Se vogliamo archiviare più file o un intero ramo del filesystem sotto UNIX si usa un comando secondario: `tar`

Archiviare più file: comando `tar` (1)

Il suo scopo è quello di aggregare i file e non quello di comprimere. Per fare ciò vengono sfruttati i meccanismi di modularità degli ambienti UNIX.

`tar`

Sintassi (semplificata): `tar [-c|-x|-t] [-z|-j|-J] [-v] [-f archive] [pathname...]`

- `-c`: crea un archivio
- `-x`: estrae un archivio
- `-t`: elenca il contenuto dell'archivio
- `-v`: mostra il progresso (modalità verbose)
- `-z`/`-j`/`-J`: comprime l'archivio con `gzip`/`bzip2`/`xz`
- `-f archive`: specifica l'archivio
- `pathname`: file e/o cartelle da comprimere

Il comando `tar` consente di aggregare file e/o cartelle in un archivio (con estenzione `.tar`). Se il `pathname` è una cartella allora viene aggiunto anche il contenuto.

Archiviare più file: comando tar (2)

Il comando `tar` accetta anche una sintassi più stringata per le opzioni: si possono omettere i trattini davanti alle opzioni e si possono, ovviamente, raggruppare.

La compressione degli archivi si ottiene facendo "passare" gli archivi `.tar` attraverso un filtro di compressione (`gzip`, `bzip2`, o `xz`) in modo esplicito (usando una pipe) oppure in modo implicito (utilizzando le opzioni `-z`,`-j` o `-J`).
Le estensioni per gli archivi compressi diventano: `.tar.gz`, `.tar.bz2` e `.tar.xz`.

I file vengono inseriti nell'archivio con il path relativo alla directory corrente.
Se non viene specificato il nome dell'archivio si lavora con lo standard input e lo standard output
Sono disponibili tante altre opzioni che permettono anche la modifica e l'aggiornamento degli archivi.

Esempi di utilizzo del comando tar

```
$ tar -c -v -f prova.tar *.tex *.pdf images/
intro.tex
shell.tex
slides.tex
slides.pdf
images/
images/dmi.jpg
images/unict.gif
..........

$ tar x -f prova.tar

$ tar cjf prova2.tar.bz2 examples/ images/ *.pdf

$ bzcat prova2.tar.bz2 | tar tv
drwxr-xr-x mario/mario        0 2005-09-20 12:28:57 examples/
drwxr-xr-x mario/mario        0 2005-09-23 14:42:28 examples/process/
-rw-r--r-- mario/mario       21 2005-09-20 12:29:27 examples/process/esempio.txt
drwxr-xr-x mario/mario        0 2005-09-20 17:40:42 examples/process/esempi/
......................

$ tar c examples/ images/ *.pdf | gzip > prova3.tar.gz
```

Gli alias

La bash da la possibilità di definire dei nomi propri in modo tale che corrispondano a sequenze arbitrarie di comandi e opzioni.

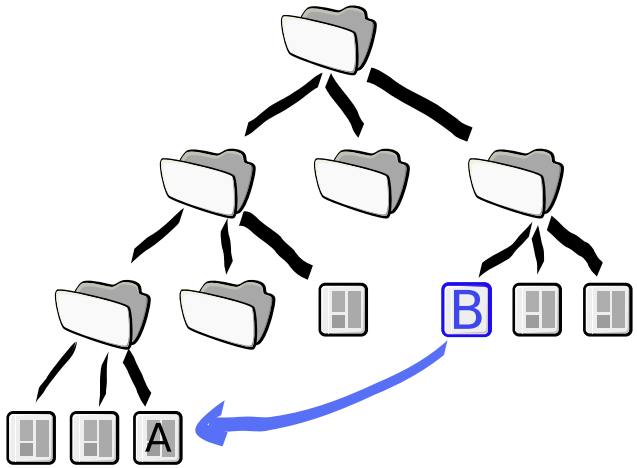
alias

Sintassi: `alias [name[=value]]`
Invocare `alias` senza parametri visualizza la lista degli alias correntemente attivi. Una invocazione del tipo `alias name` visualizza l'associazione attuale (se ne esiste già una). Per rimuovere un alias si utilizza il comando `unalias name`.

```
$ alias ll='ls -l'

$ ll /
drwxr-xr-x    2 root root  4096 2005-09-23 08:27 bin
drwxr-xr-x    3 root root  4096 2005-09-23 09:26 boot
drwxr-xr-x   11 root root 13520 2005-09-23 12:30 dev
..............

$ alias ls='ls --color'

$ alias rm='rm -i'

$ unalias rm ls
```

I link nel filesystem

I filesystem UNIX mettono a disposizione un meccanismo molto potente che permette di riferirsi ad un dato oggetto nel filesystem con più di un riferimento (o link).
Dato un file A nel filesystem, è possibile collocare in un punto qualunque del filesystem un link B che "punti" ad A. A livello di applicazione lavorare con A o lavorare con B hanno gli stessi effetti: in effetti operiamo sul medesimo file. Si hanno due nomi per indicare lo stesso oggetto.
Esistono due tipi di link: simbolici e fisici



I link nel filesystem: link simbolici

I link simbolici (detti anche soft link) sono dei file speciali che contengono al loro interno il riferimento al file a cui puntano.
In effetti sono dei piccoli file di testo contenente il path assoluto o relativo (alla directory in cui si trova il link simbolico) per raggiungere il file a cui si riferiscono (il file target).

```
$ ls -l file_?
-rw-r--r--  1 mario mario 2389 2005-09-23 18:50 file_A
lrwxrwxrwx  1 mario mario    6 2005-09-23 18:48 file_B -> file_A
```

Una invocazione del comando `ls -l` con tutti i dettagli sui file rivela la natura del link simbolico: un flag `l` vicino ai diritti indica che si tratta di un link simbolico; vicino al nome del link simbolico viene indicato il nome del file target.

Se viene rimosso il soft link, il file target rimane inalterato. Se si cancella il file target, il soft link diventa inconsistente.

I link nel filesystem: link fisici (2)

I link fisici (detti anche hard link) hanno una natura diversa ed agiscono ad un livello del filesystem più basso.
Senza scendere nei dettagli (lo vedremo meglio più avanti nel corso), i filesystem UNIX rappresentano ogni oggetto (file o cartella) con un inode memorizzato in una parte del disco. Se una cartella contiene quel file, avrà un riferimento all'indirizzo dell'inode che memorizza il file (come se fosse un puntatore ad un oggetto).

Un hard link agisce creando, nella cartella che lo contiene, un secondo puntatore al medesimo inode. Ci saranno due puntatori allo stesso inode.

Dato un file e creato un hard link ad esso, a posteriori è impossibile distinguere qual'era il nome originale e quale il link quindi si ha una trasparenza totale a livello di applicazione.

I link nel filesystem: link fisici (2)

Per gestire gli hard link, il filesystem UNIX mantiene per ogni inode un contatore del numero di link fisici che puntano ad esso.

Per sapere quanti hard link puntano allo stesso file si può usare il comando `ls`:

```
$ ls -l file_?
-rw-r--r--  2 mario mario 2389 2005-09-23 18:50 file_A
-rw-r--r--  2 mario mario 2389 2005-09-23 18:50 file_B
```

Il numero degli hard link è riportato subito dopo i permessi di accesso.

Poiché gli hard link lavorano a livello di inode, si possono creare solo tra file del medesimo filesystem.
Con i link simbolici è invece possibile creare link a file che si trovano in filesystem diversi.
Per implementare gli hard link è necessario che il filesystem lo supporti: tutti i filesystem UNIX (ext2, ext3, reiserfs, nfs, ...) lo supportano, quelli di Windows (VFAT,NTFS) no.

I link nel filesystem: link fisici (3)

Quando si cancella un hard link viene innescato il medesimo meccanismo utilizzato quando si cancella un semplice file:

- il sistema operativo controlla il contatore dei link all'interno dell'inode del file che si vuole cancellare;
- se tale contatore è uguale ad 1, allora viene cancellato sia il link (nella directory) che l'inode stesso (compreso il contenuto del file);
- se il contatore è strettamente maggiore di 1, allora esso viene decrementato e viene cancellato solo il riferimento all'inode e quest'ultimo viene preservato.

Si possono creare link simbolici di directory ma non è possibile fare altrettanto con i link fisici. Altrimenti si creerebbero dei loop nella gerarchia del filesystem che creerebbero problemi con i programmi che lo esplorano. Ciò non è un problema con i soft link poiché sono distinguibili dalle directory vere.

I link nel filesystem: comandi

`ln`

Sintassi: `ln [-s] target... [linkpathname]`

- `-s`: crea un link simbolico invece che uno fisico
- `target`: file o directory a cui il link farà riferimento
- `linkpathname`: pathname del link

Il comando `ln` consente di creare link tra i file e le directory presenti nel filesystem. I link vengono creati utilizzando il parametro `linkpathname` e fanno riferimento al `target` specificato. Se si specificano più `target`, `linkpathname` deve essere una directory: verranno creati tutti link con gli stessi nomi dei target.
Se si omette il parametro `linkpathname` il link viene creato nella directory corrente usando lo stesso nome del target.

Modalità di esecuzione: *simple command execution* (1)

Esistono varie modalità di esecuzione dei comandi sotto una shell. La più semplice consiste nell'invocazione di un singolo comando (con relativi parametri).

Ogni comando restituisce un exit status che rappresenta l'esito della computazione del comando stesso. Mediante l'exit status è possibile controllare la buona riuscita di un comando.

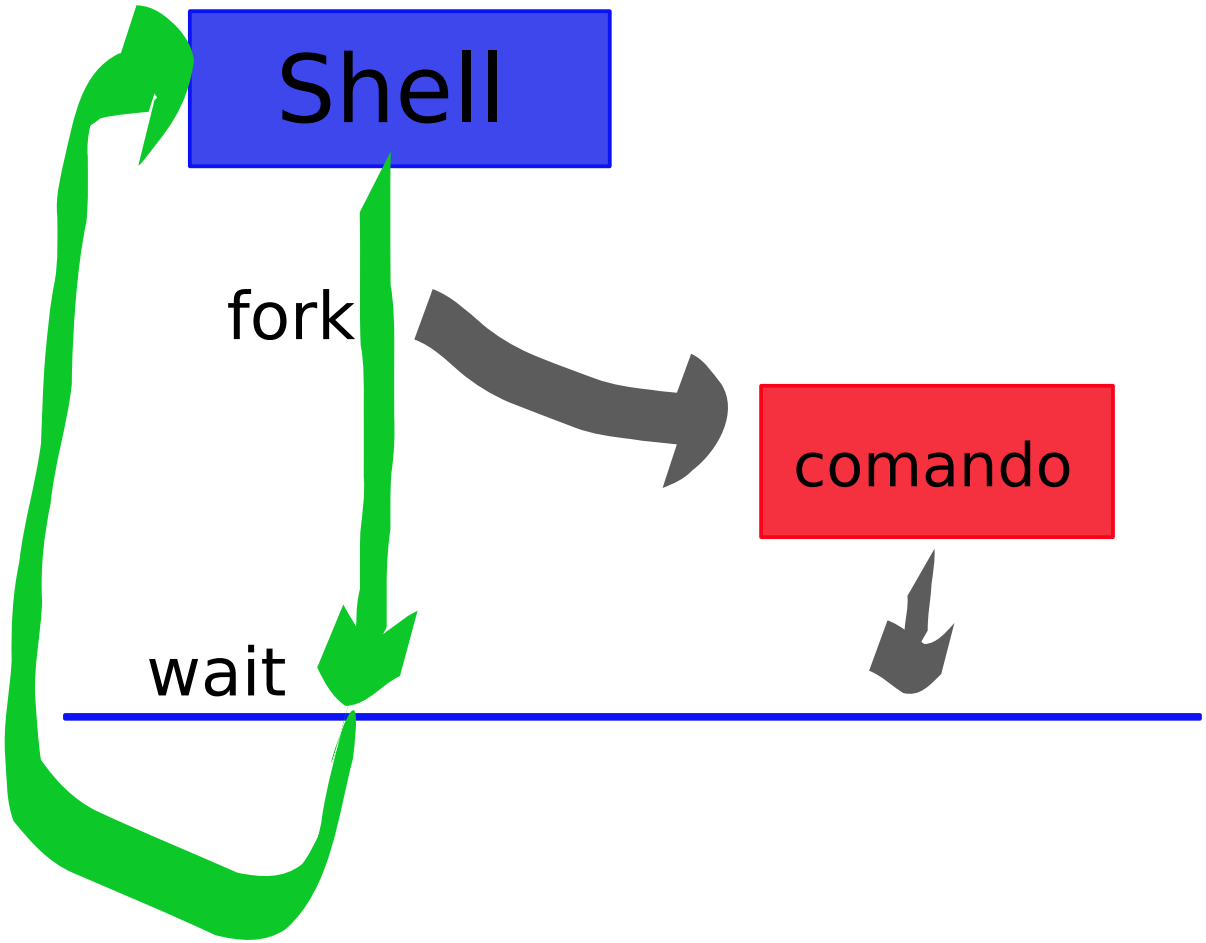
L'exit status è un intero:

`0`: esecuzione riuscita con successo;

`n > 0`: esecuzione fallita;

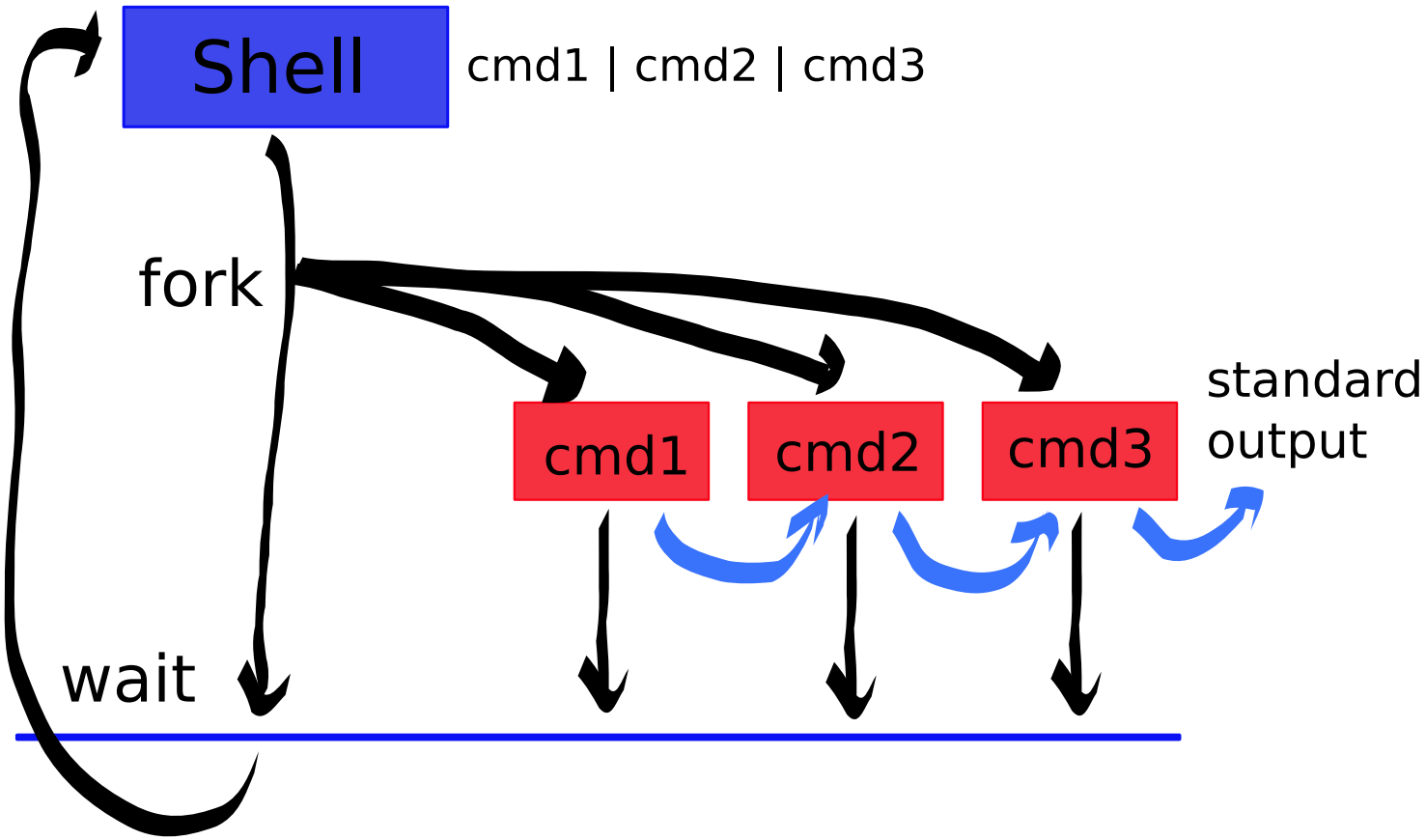
L'esecuzione di un comando da shell, la sospende temporaneamente fino alla terminazione del comando stesso.

Modalità di esecuzione: *simple command execution* (2)



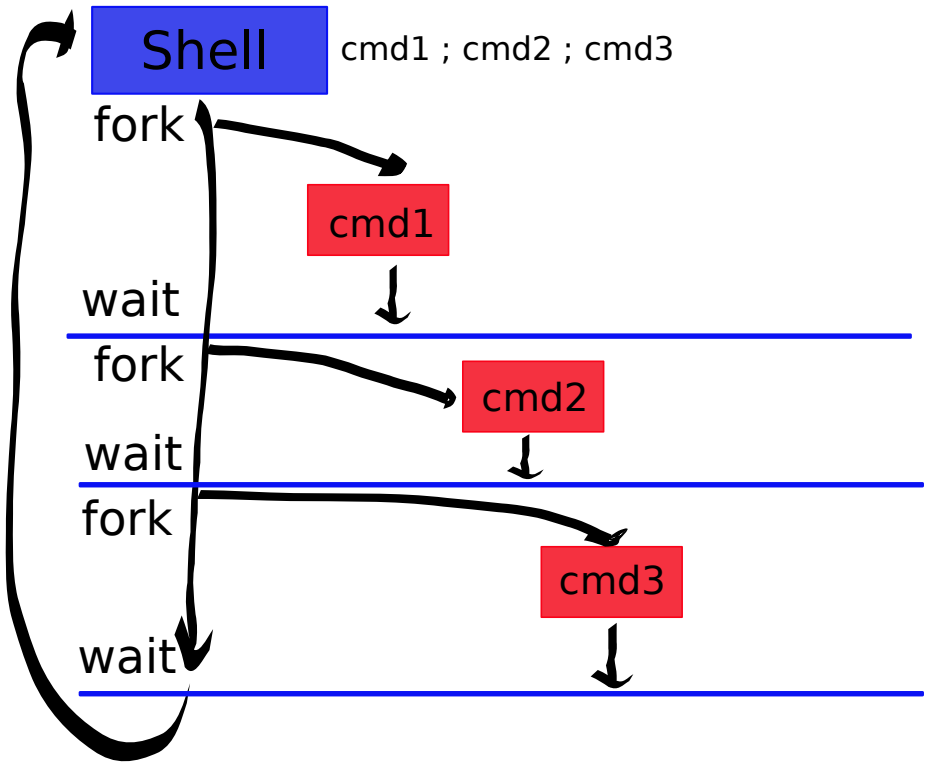
Modalità di esecuzione: pipeline

L'abbiamo già spiegata prima: una cascata di processi in cui ognuno riceve l'output del precedente come input. L'output della pipeline corrisponde all'output dell'ultimo processo.



Modalità di esecuzione: *list of command* (1)

La lista di comandi permette di eseguire una sequenza di comandi come se fossero un solo comando.
La sintassi per l'invocazione è: `commando1 [; comando2...]`
L'exit status della sequenza è uguale a quello dell'ultimo comando. Non c'è alcuno scambio di input/output tra i processi. La shell attende che l'ultimo



Modalità di esecuzione: *list of command* (2)

Nelle sequenze di comandi, oltre al separatore '`;`', è possibile utilizzare anche gli operatori condizionali `&&` e `||`.
La semantica degli operatori è la seguente:

- `c1 && c2` (*and list*): `c2` viene eseguito se e solo se l'exit status di `c1` è 0
- `c1 || c2` (*or list*): `c2` viene eseguito se e solo se l'exit status di `c1` è diverso da 0

L'exit status di una lista condizionata è uguale all'exit status dell'ultimo comando eseguito (che non è necessariamente l'ultimo comando della lista).

```
$ cat fileinesistente && echo fatto
cat: fileinesistente: No such file or directory

$ cat fileinesistente || echo errore
cat: fileinesistente: No such file or directory
errore

$ false && comando_inesistente_che_non_verra_mai_eseguito
```

Modalità di esecuzione: *asynchronous execution*

Utilizzando la sintassi `comando &` (notare la e-commerciale alla fine), il comando viene mandato in esecuzione asincrona (o più semplicemente in background).
In questo caso la shell manda in esecuzione il processo figlio per eseguire il comando e continua la sua attività (non attende la fine del task).

Di default, i processi in esecuzione asincrona prendono il loro input da `/dev/null` (se non specificato).
Un processo in `foreground` può essere mandato in background interattivamente utilizzando la combinazione di tasti `CTRL-Z`. In effetti il processo viene anche sospeso (non continua la sua esecuzione), viene messo in pausa.

Controllo dei *jobs*

La shell associa ad ogni comando semplice (o pipeline) uno job. Ogni job ha un numero che lo identifica univocamente in un dato istante.

Quando un comando viene eseguito in modo asincrono, la shell da un output del tipo:
`[jobnumber]  PID`
dove `jobnumber` è il numero dello job eseguito in background e `PID` è l'identificativo di processo che porta effettivamente avanti il job.

Comandi disponibili:

- `jobs`: visualizza i jobs correntemente attivi ed il loro stato;
- `bg`: manda in background l'esecuzione di uno job;
- `fg`: porta in foreground l'esecuzione di uno job;
- `kill`: invia un segnale ad uno job.

A proposito di `kill`: con `kill -# PID` si invia il segnale numero `#` al processo con con identificativo `PID`.
Tra quelli standard: 3 QUIT e 9 KILL.

Raggruppare i comandi

E' possibile i comandi in modo che vengano considerati come una singola unità utilizzando le seguenti notazioni:
`( list )` oppure `{ list ; }`
Tutte le redirezioni applicate ai raggruppamenti di comandi vengono utilizzate per redirigere tutto l'output dei comandi in un unico stream.

```
$ true && echo -n Supercali; echo -n fragilisti ; echo  chespiralidoso
Supercalifragilistichespiralidoso

$ false && echo -n Supercali; echo -n fragilisti ; echo  chespiralidoso
fragilistichespiralidoso

$ true && (echo -n Super; echo -n fragilisti ; echo  chespiralidoso)
Supercalifragilistichespiralidoso

$ false && (echo -n Super; echo -n fragilisti ; echo  chespiralidoso)
```

Le variabili

Una variabile è un oggetto che memorizza un valore. Le variabili sono identificate da stringhe alfanumeriche che iniziano con un carattere.
L'assegnamento di un valore ad una variabile viene effettuata mediante l operatore di assegnamento `=`. La sintassi esatta è `nome_variabile=valore`, dove prima e dopo del carattere `=` NON ci deve essere nessuno spazio.
L'assegnamento ad una variabile resta valido solamente per la shell corrente. Se si vuole propagare l'assegnamento anche ai processi figli (come i comandi eseguiti nella shell) bisogna utilizzare il comando interno `export` specificando come parametri le variabili da esportare.
Il valore nullo è un valido assegnamento per una variabile di shell. Per annullare un assegnamento si usa il comando `unset`.
Con il comando `set` è possibile visualizzare tutte le variabili correntemente definite in una shell (comprese le variabili d'ambiente).

Espansione delle variabili

Quando la shell esegue un comando effettua delle espansioni della riga di comando, tra queste abbiamo le espansioni per i metacaratteri (che abbiamo già visto) e le espansioni delle variabili.
Mettendo il carattere `$` davanti al nome di una variabile definita, questa viene espansa con il suo valore.
Quindi la sintassi è: `$nome_variabile`, che rappresenta una semplificazione della sintassi generale `${nome_variabile}` che risulta necessaria in alcuni casi.

Variabili speciali e di shell

La shell mette a disposizione alcune variabili speciali:

- `$?`: exit status del comando più recente eseguito in foreground;
- `$$`: PID della shell corrente;
- `$!`: PID del più recente job eseguito in background.

Così come alcune variabili di shell predefinite:

- `$USER`: utente corrente;
- `$HOME`: la home directory associata con l'utente;
- `$PATH`: la lista delle directory che deve essere ispezionata per trovare i comandi eseguibili;
- `$PS1`: la stringa corrispondente al prompt corrente.

Command substitution

Mediante la command substitution è possibile sostituire l'output di un comando con il comando stesso (parlando in termini di espansione della riga di comando).

La sintassi è: `$(comando)`

Risulta molto utile per inizializzare le variabili.

```
$ ls | wc -w
15

$ PAROLE=$(ls | wc -w)

$ echo parole conteggiate: $PAROLE
parole conteggiate: 15
```

Quoting

Il meccanismo del quoting serve ad eliminare il metasignificato ad alcuni dei caratteri che vengono usati per altri scopi.
Il quoting può essere di tre tipi:

- \ (*backslash*): elimina il metasignificato del carattere seguente;
- ' ' (*single quote*): elimina il metasignificato da tutti i caratteri contenuti al suo interno;
- " " (*double quote*): elimina il metasignificato da tutti i caratteri contenuti al suo interno ad eccezione di $, ` e \.

```
$ echo $PATH
/home/mario/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
$ echo \$PATH
$PATH

$ echo "Il mio nome è $USER."
Il mio nome è pippo.

$ echo 'Il mio nome è $USER.'
Il mio nome è $USER.

$ echo "Il mio nome è `whoami` e ho 5\$."
Il mio nome è pippo e ho 5$.
```

Dove ho messo quei file!?

`find`

Sintassi: `find [pathname...] [expression]`

- `pathname`: percorso in cui cercare (ricorsivamente) i file da esaminare
- `expression`: specifica le regole con cui selezionare i file; può essere una fra le seguenti:
  - `opzione`: modifica il comportamento della ricerca; per esempio `-mount`;
  - `condizione`: condizioni da verificare; ad esempio `-name modello`, `-user utente`, `-group gruppo`, `-type c`, ...
    E' possibile usare anche operatori logici tipo: `-not expr`, `expr1 -or expr2`, ...
  - `azione`: specifica cosa fare quando trova un file; ad esempio `-exec comando {}` , `-print`, ...

Ricerca ricorsivamente all'interno dei `pathname` specificati dei file che soddisfino le espressioni `expression` date. Man mano che i file vengono trovati i nomi vengono stampati a video o altro se specificato.

Le opzioni sono tante, per ulteriori dettagli consultare `man find`.

Esempi di utilizzo del comando `find`

```
$ find . -name '*.sh' -print
./script.sh
./prova.sh
./conta.sh
./cartella/esercizio/eseguimi.sh

$ find /home/mario -name '*.bak' -exec rm {} \;

$ find /etc/ -type d -print
/etc/
/etc/mkinitrd
/etc/network/if-post-down.d
/etc/network/if-pre-up.d
/etc/network/if-up.d
/etc/network/if-down.d
/etc/network/run
/etc/default
/etc/skel
...........
```

Facciamo una cernita: il comando grep

grep (General Regular Expression Parser)

Sintassi (semp.): `grep [-i][-l][-n][-v][-w] pattern [filename...]`

- `-i`: ignora le differenze tra maiuscole e minuscole
- `-l`: fornisce la lista dei file che contengono il `pattern`
- `-n`: le linee dell'output sono precedute dal numero di linea
- `-v`: stampa solo le linee che `non` contengono il `pattern`
- `-w`: vengono restituite solo le linee che contengono il pattern/stringa come parola completa
- `pattern`: espressione da ricercare
- `filename`: file da setacciare

Cerca all'interno delle righe dei file specificati con `filename` le righe che contengono (o meno, a secondo delle opzioni) il `pattern` specificato. Il `pattern` può essere una semplice stringa o una regular expression.

Espressioni Regolari

Attraverso le espressioni regolari è possibile specificare dei *pattern* più complessi della semplice stringa contenuta.

metacarattere	tipo	significato
`^`	**basic**	inizio della linea
`$`	**basic**	fine della linea
`.`	**basic**	un singolo carattere (qualsiasi)
`[str]`	**basic**	un qualunque carattere in `str`
`[^str]`	**basic**	un qualunque carattere **non** in `str`
`[a-z]`	**basic**	un qualunque carattere tra `a` e `z`
`\`	**basic**	inibisce l'interpretazione del carattere successivo
`*`	**basic**	zero o più ripetizioni dell'elemento precedente
`+`	**ext**	una o più ripetizioni dell'elemento precedente
`?`	**ext**	zero o una ripetizione dell'elemento precedente

Esempi di utilizzo con grep e varianti

- `grep -F rossi /etc/passwd`
  fornisce in output le linee del file `/etc/passwd` che contengono la stringa fissata `rossi`
- `grep -G -nv '[agt]+' relazione.txt`
  fornisce in output le linee del file `relazione.txt` che non contengono stringhe composte dai caratteri `a`, `g`, `t` (ogni linea è preceduta dal suo numero)
- `grep -w print *.c`
  fornisce in output le linee di tutti i file con estensione `c` che contengono la parola intera `print`
- `ls -al .  | grep '^d.......w.'`
  fornisce in output le sottodirectory della directory corrente che sono modificabili da tutti gli utenti del sistema
- `grep -G '[a-c]+z' doc.txt`
  fornisce in output le linee del file `doc.txt` che contengono una stringa che ha un prefisso di lunghezza non nulla, costituito solo da lettere `a`, `b`, `c`, seguito da una `z`

Cos'è uno script?

Uno script consiste in un insieme strutturato di comandi shell con strutture di controllo molto simili a quelle dei normali programmi scritti in linguaggio C.
Si tratta di strutture di programmazione molto semplici e prettamente imperative.

Praticamente, si tratta di un file di testo contenente le istruzioni da eseguire. Uno script `programma` può essere eseguito utilizzando la seguente sintassi: `bash programma`.
Oppure, si può attivare il flag eseguibile sul file di testo (ad esempio, con un `chmod +x programma`) ed invocarlo con `./programma` (il `./` è necessario nel caso in cui lo script sia contenuto nella cartella corrente e `PATH` non la contiene).

Come iniziare uno script e i commenti

Convenzionalmente ogni script dovrebbe iniziare con una prima riga del tipo `#!/bin/bash`, dove viene indicato interprete necessario per eseguire lo script. La stessa va applicata nel caso di una shell differente (`#!/bin/sh` o `#!/bin/tcsh`) o di un interprete di un linguaggio (`#!/usr/bin/perl` o `#!/usr/bin/python`).

Curiosità

In realtà i due caratteri iniziali non sono casuali, si tratta di particolari magic number che aiutano il sistema ad identificare velocemente di che tipo sono i file. In questo caso i magic number per gli script da interpretare sono `0x23 0x21`. Esistono dei magic number per molti tipi di file (PDF, GIF, JPG, ecc.).

Se si omette l'intestazione i risultati non sono garantiti.

Inoltre il carattere cancelletto # ha la funzione di iniziare un commento: tutto ciò che segue fino alla fine della riga viene ignorato dall'interprete script.