Introduzione alla Shell e agli Script

(slide in versione estesa)

Mario Di Raimondo

Associazione Software Libero Ragusa (SoLiRa)

linux@evening 2010

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.

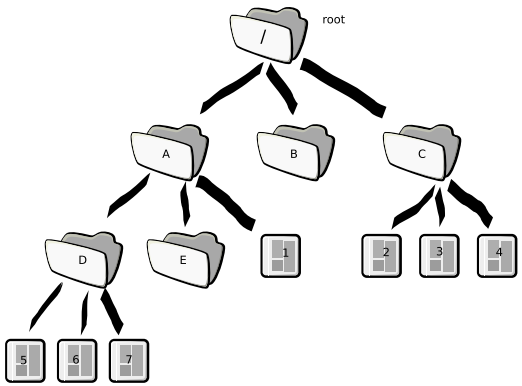
Il filesystem

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

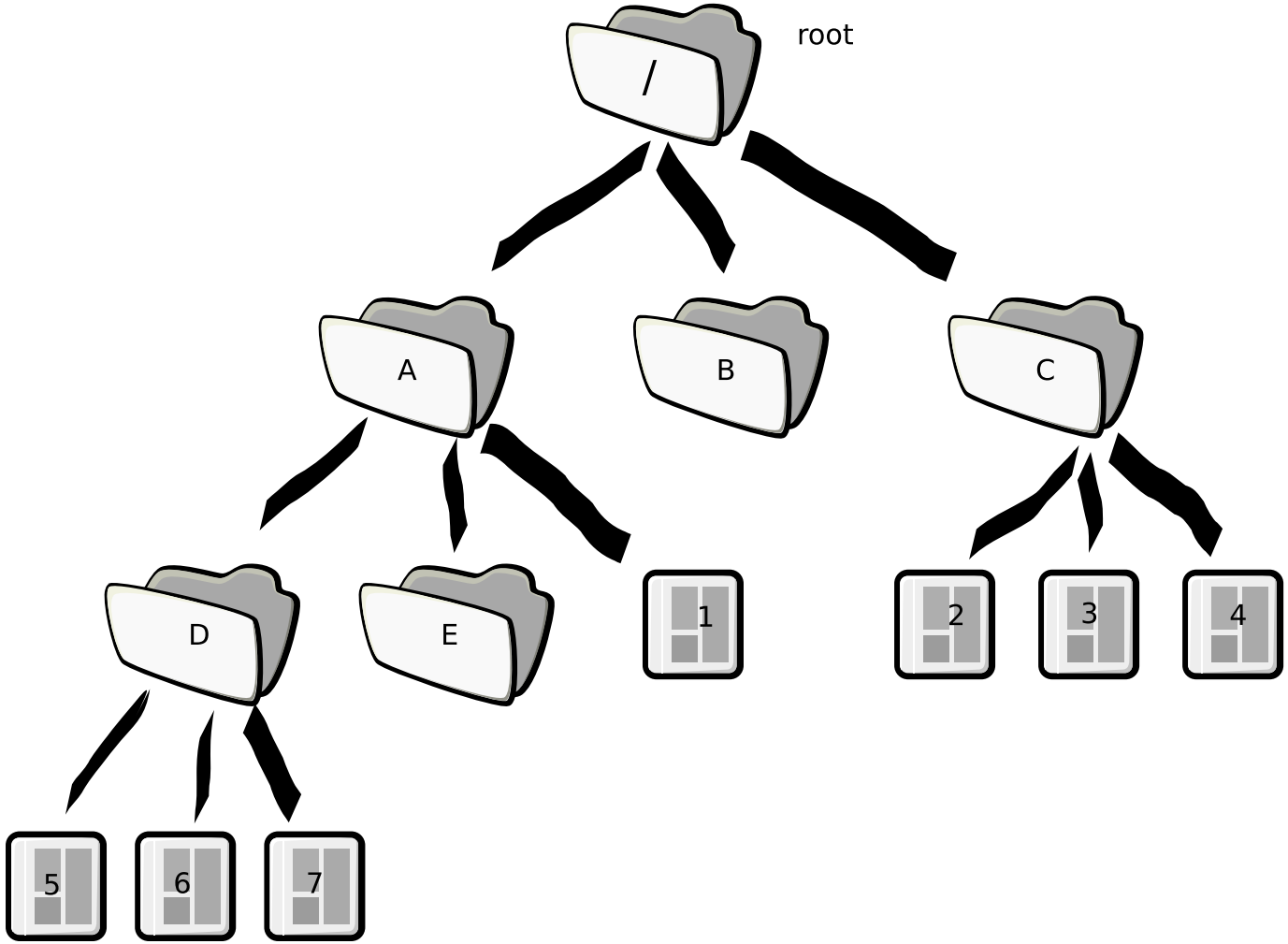
Il filesystem contiene i seguenti tipi di oggetti:

- 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:

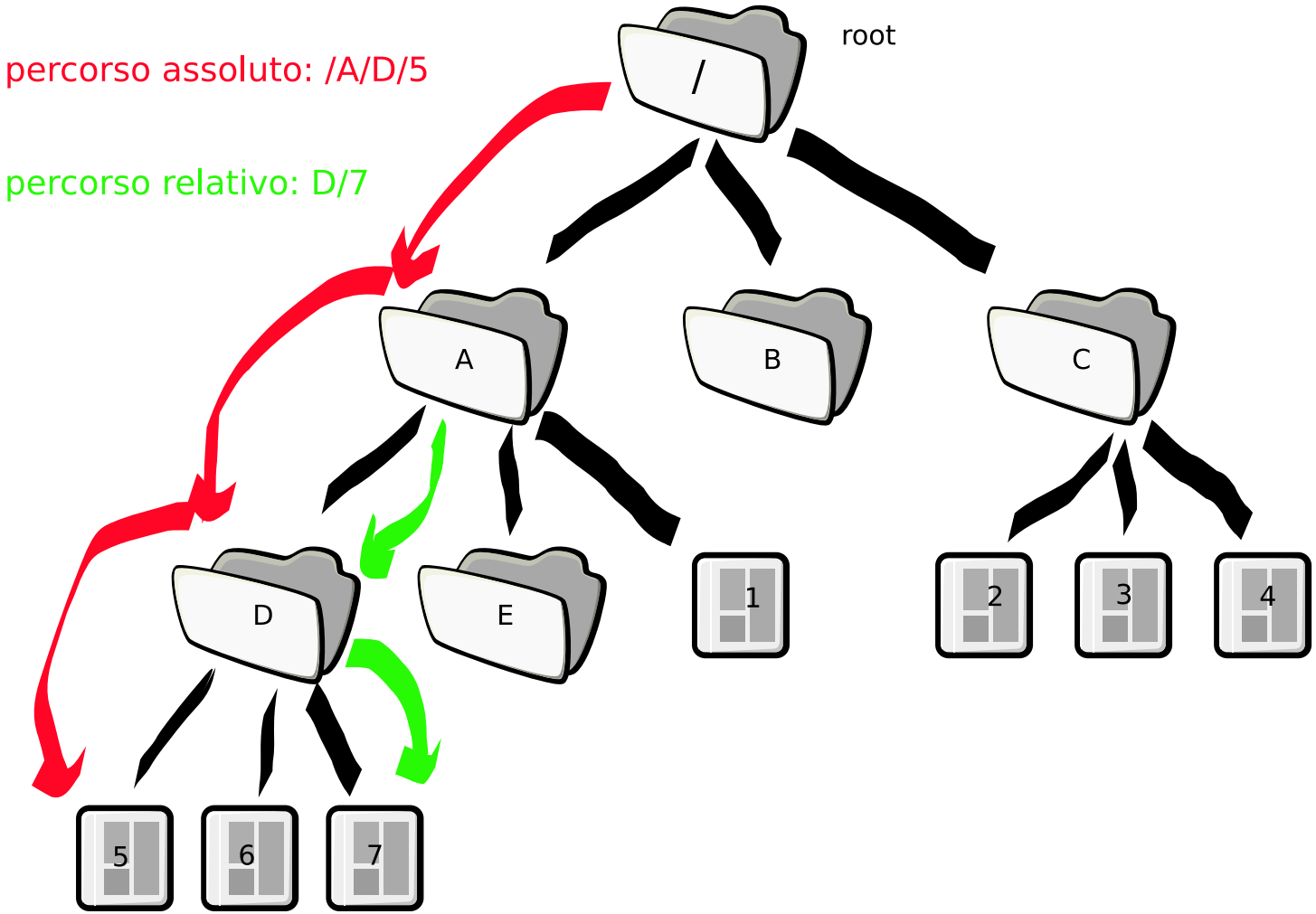
Percorso assoluto

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

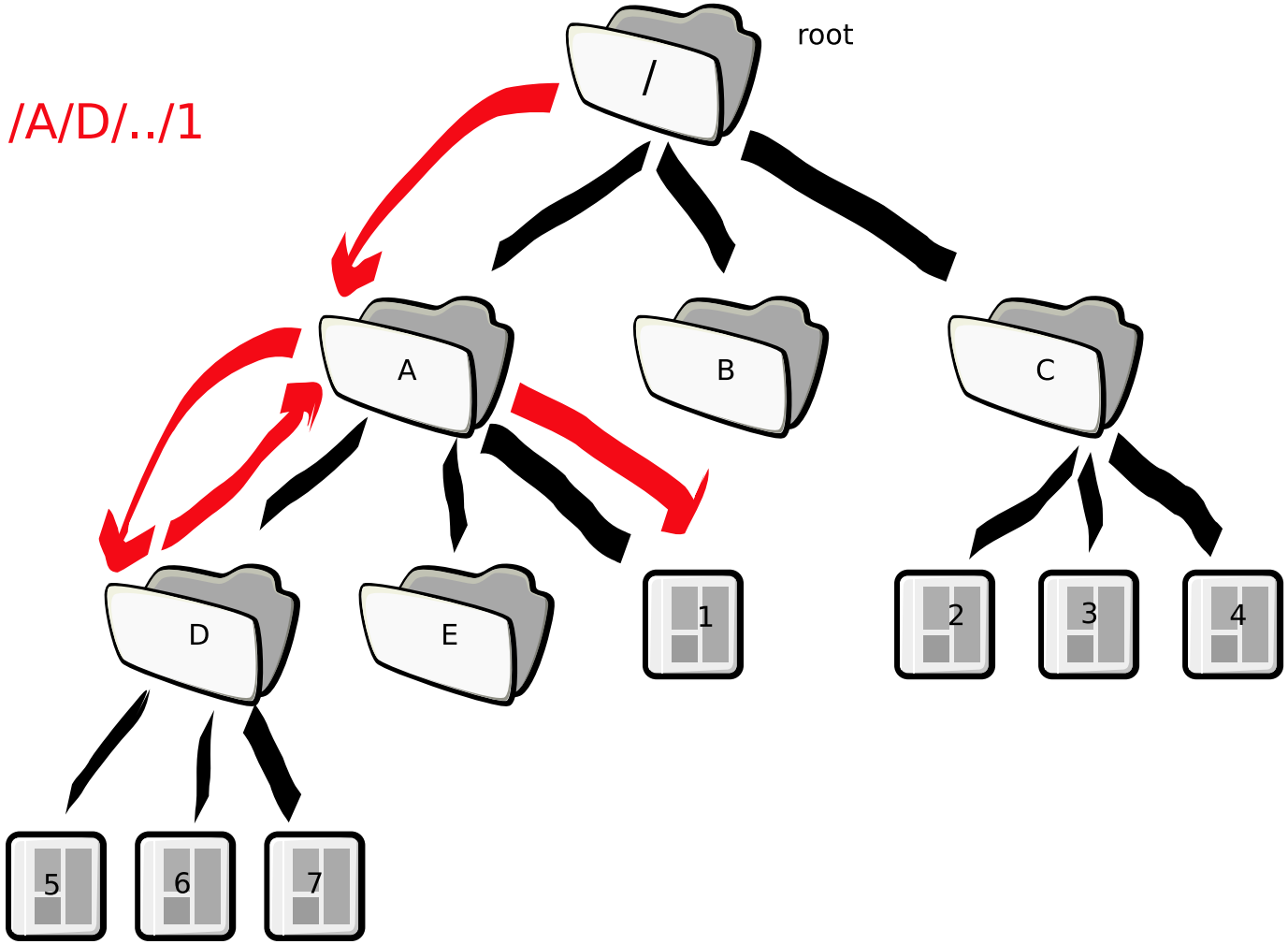
In effetti il percorso assoluto di un file è il suo percorso relativo rispetto alla root.

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.

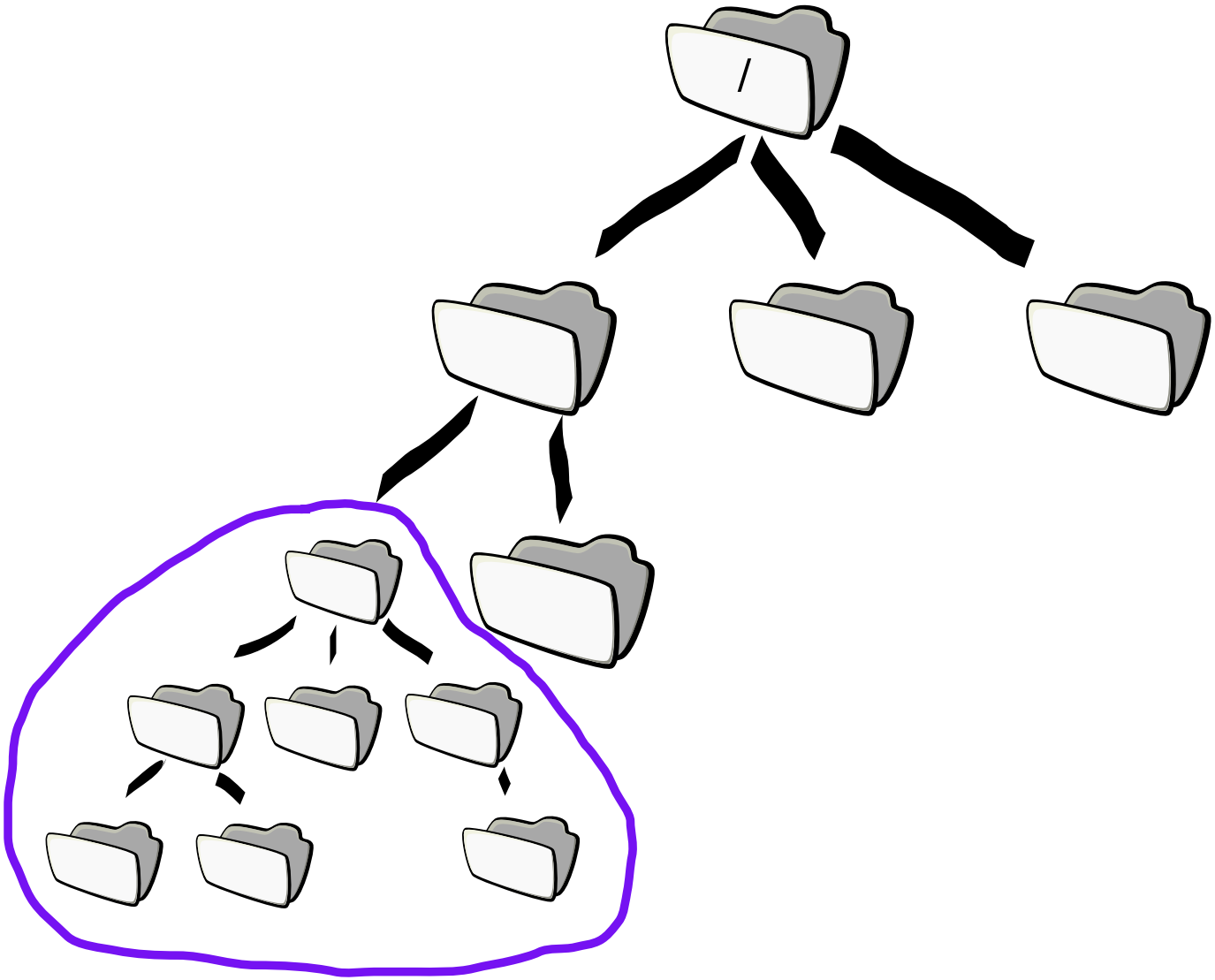
Esempi: dischi ATA (`/dev/hda`,`/dev/hdb`,...), partizioni all'interno dei dischi (`/dev/hda1`,`/dev/hda2`,...), floppy disk (`/dev/fd0`), terminali testuali (`/dev/tty1`), schede audio (`/dev/dsp`).

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.

La shell bash

Esistono varie shell sui sistemi UNIX:

- sh: Bourne Shell
- bash: Bourne Again Shell
- csh: C Shell
- tcsh: Teach C Shell
- ksh: Korn Shell

Noi utilizzeremo come riferimento la bash (poiché è la più diffusa e completa), ma la maggior parte delle nozioni introdotte in questo corso si potranno impiegare anche su altri sistemi.
Si trova sulla maggior parte dei sistemi UNIX moderni (volendo si può installare anche su Windows!) e fa parte del progetto GNU.

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`.

- `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.

History list (1)

La history list è uno strumento messo a disposizione dalla shell bash che consente di evitare all'utente di digitare più volte gli stessi comandi:

- bash memorizza, di default, nella history gli ultimi 500 comandi inseriti dall'utente;
- la lista viene memorizzata nel file nascosto `.bash_history` nella home directory dell'utente al momento del logout e viene riletta in fase di login;
- il comando `history` visualizza tutta la lista:

```
$ history
  .........
  511 pwd
  512 ls -al
  513 cd /etc
  514 more passwd
  515 history
```

- ogni riga prodotta è detto evento ed è preceduto da un numero di evento;
- si può rieseguire l'ultimo comando usando la sintassi `!!`.

History list (2)

- si può reinvocare un comando usando la sintassi `!###`, dove `###` è il numero di evento:

```
$ !515
history
  .........
  512 ls -al
  513 cd /etc
  514 more passwd
  515 history
  516 history
```

- in modo molto più utile è possibile scorrere la lista degli eventi già eseguiti con i tasti freccia su e giù;
- è possibile anche effettuare ricerche all'interno dell'history: prima si avvia la ricerca utilizzando la combinazione di tasti `CTRL-R` e poi si inizia a digitare la stringa da ricercare; la ricerca avviene dall'evento più recente in su e il risultato parziale viene visualizzato a schermo:

```
$
(reverse-i-search)‘pass’: more passwd
```

Command completion (1)

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 indica la pressione del tasto tabulazione).

```
$ pass<tab>
```

La pressione del tasto 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 tabulazione;
- alla seconda pressione del tasto tabulazione la shell visualizza una lista delle possibili alternative;
- digitando ulteriori caratteri, alla successiva pressione del tasto tabulazione, la lunghezza della lista diminuirà fino ad individuare un unico comando.

Command completion (2)

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

```
$ ls /etc/p<tab><tab>
pam.conf          papersize         php4/             profile
pam.d/            passwd            pmount.allow      protocols
pango/            pcmcia/           ppp/              proxychains.conf
paper.config      perl/             printcap          python2.3/
$ ls /etc/pa<tab><tab>
pam.conf      pango/        papersize
pam.d/        paper.config  passwd
$ ls /etc/pas<tab>
/etc/passwd
```

In questo caso alla prima doppia pressione del tasto tabulazione, la shell presenta tre possibili alternative; digitando una a e premendo due volte il tasto tabulazione, la shell ha una quantità di informazione sufficiente per determinare in modo univoco il completamento del nome di file.
Il completamento funziona anche con i nomi di directory e risulta molto utile per attraversare velocemente il filesystem.

Esiste anche una versione più avanzata (opzionale) della bash completion che permette di completare automaticamente anche le opzioni dei comandi e di contestualizzare il completamento dei file passati per argomento.

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 (magari ne discuteremo in altra sede...);
- 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 (anche questo non sarà discusso in questa sede).

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;
- { }: 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
```

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 `mkdir` 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 `cp` 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.

Questi due comandi sono prettamente interattivi.

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 [-b] [-f] [-n] [-r] [-o file] [-t s] [-k s1,s2] [pathname]`

- `-b`: ignora eventuali spazi presenti nelle chiavi di ordinamento
- `-f`: ignora le distinzioni tra maiuscole e minuscole
- `-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 `s1` ad `s2`-1 come campi di 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 3,4
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 3,4 -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`

Taglia...

cut

Sintassi (semplificata): `cut [-c list] [-f list] [-d s] [pathname...]`

- `-c list`: stampa solo i caratteri indicati da `list`
- `-f list`: stampa solo i campi indicati da `list`
- `-d s`: quando si usa l'opzione `-f`, utilizza `s` come delimitatore
- `pathname`: file da analizzare

Il comando `cut` estrae degli elementi (caratteri o campi) da ogni riga del suo input. Il separatore di default è la tabulazione.
Le liste `list` sono sequenze, separate da virgola, indicanti i caratteri o campi da selezionare. Si possono indicare anche intervalli chiusi o aperti del tipo `n-m`, `-m`, `n-`. La numerazione inizia da 1.

... e Incolla

paste

Sintassi (semplificata): `paste [-d delimiters] [pathname...]`

- `-d delimiters`: prende di volta in volta i delimitatori da usare da `delimiters`
- `pathname...`: file giugere

Il comando `paste` unisce i file passati riga per riga, separandole con un delimitatore (di default la tabulazione).

Esempi di utilizzo di cut e di paste

```
$ echo "Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura" |
  cut -d" " -f 1-2,5,11-
Nel mezzo di una selva oscura

$ echo "Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura" |
  cut -c 11,16,20,34,50
dante

$ date
gio set 22 17:13:01 CEST 2005
$ date | cut -d" " -f4 | cut -d: -f -2
17:14

$ cat /etc/passwd | cut -d: -f5
root
daemon
bin
sys
.......

$ cat /etc/passwd | cut -d: -f1 > nomi.txt
$ cut -d: -f3 /etc/passwd | paste -d, nomi.txt -
root,0
daemon,1
bin,2
sys,3
sync,4
.......
```

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----- ⟷ u=rw,g=r,o-rwx
rw-rw-rw- + u+x,o-rw ⟶ rwxrw----
rw-r--r-- + a-r,u+r ⟶ rw-------
rw-r--r-- + +x ⟶ 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: `chown [-R] group [pathname...]`

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

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`.

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` e `bzip2`.

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] [-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`: comprime l'archivio con `gzip`/`bzip2`
- `-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` o `bzip2`) in modo esplicito (usando una pipe) oppure in modo implicito (utilizzando le opzioni `-z` o `-j`).
Le estensioni per gli archivi compressi diventano: `.tar.gz` e `.tar.bz2`.

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
```

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).

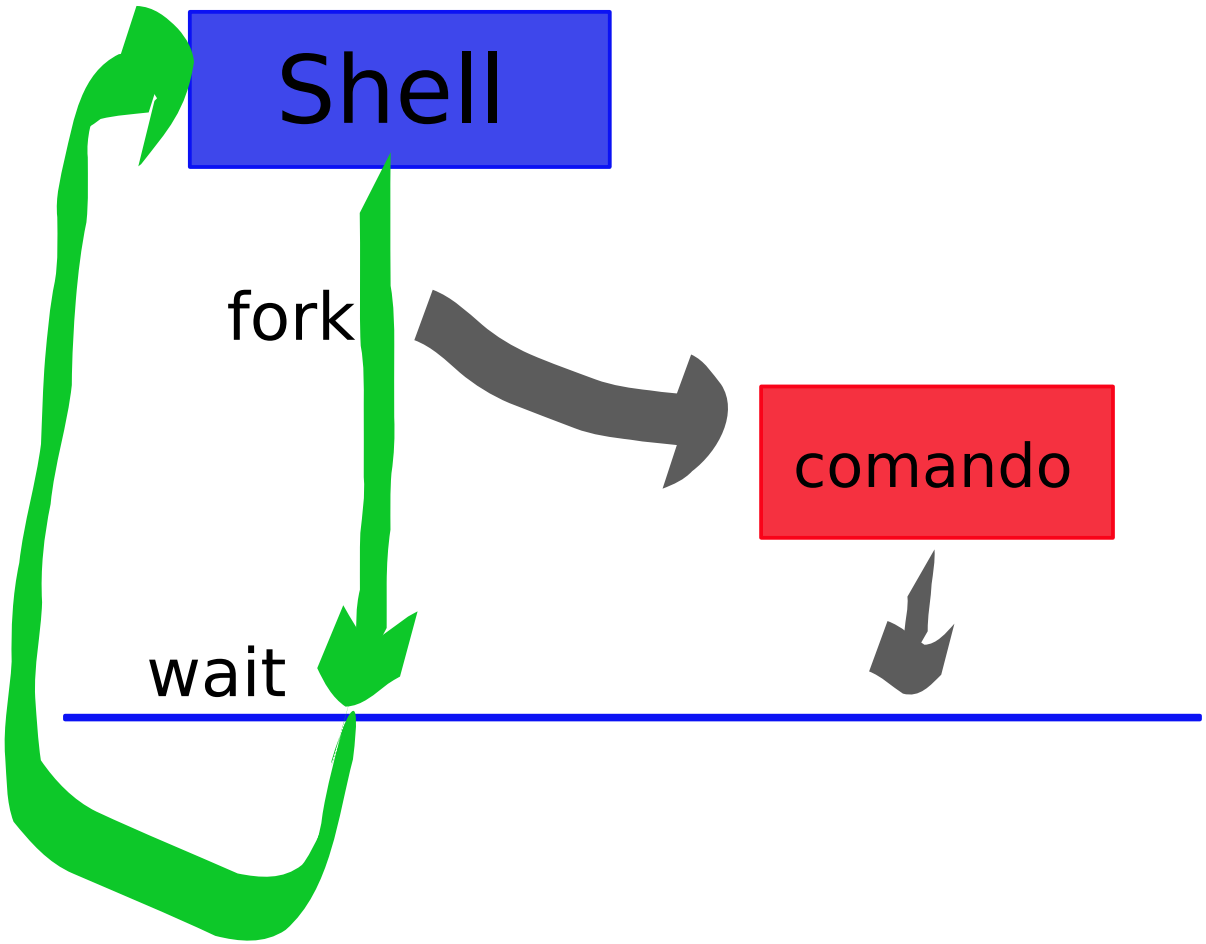
Ogni comando restituisce un exit status che rappresenta l'esito della computazione del comando stesso. Mediante l'exit status è possibile controllare la buona riuscita di un comando.

L'exit status è un intero:

- `0`: esecuzione riuscita con successo;
- `n > 0`: esecuzione fallita;
- `128 + n`: esecuzione terminata a seguito di un segnale `n`.

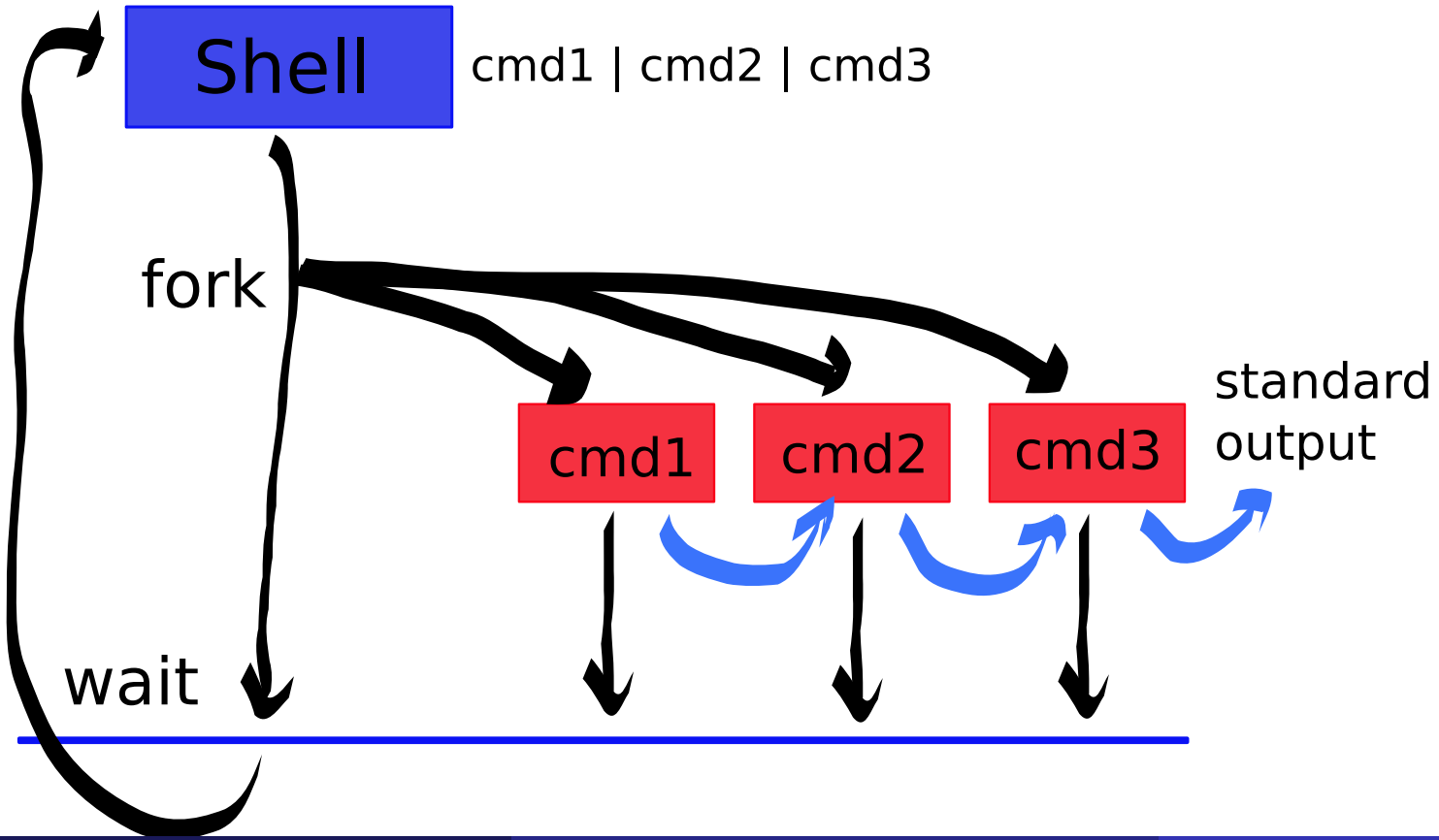
L'esecuzione di un comando da shell, la sospende temporaneamente fino alla terminazione del comando stesso.

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



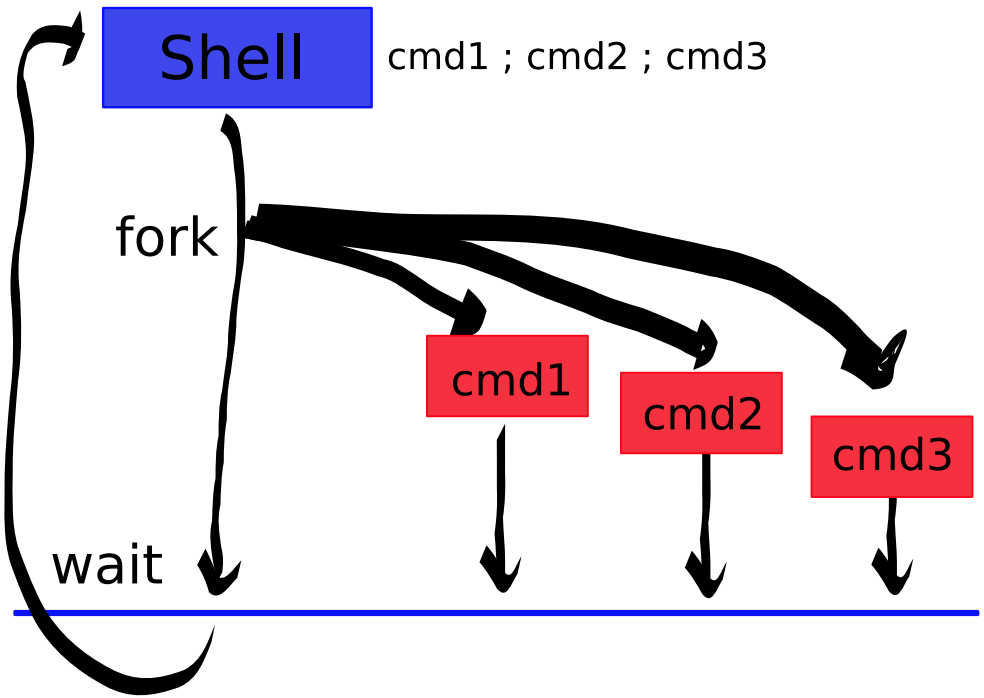
Modalità di esecuzione: pipeline

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).

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 degli alias (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:

- `$?`: exit status del comando più recente eseguito in foreground;
- `$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;

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‘` (con le apici inverse ottenibili utilizzando il tasto ALT GR e l'apice normale ').

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.

Esistono due varianti di `grep`: `fgrep` (Fixed General Regular Expression Parser) e `egrep` (Extended General Regular Expression Parser).

Volendo si possono ottenere i medesimi risultati con le opzioni `-F` e `-G` di `grep`.

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**	inizio di una parola
`\>`	**basic**	fine di una parola
`.`	**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

- `fgrep rossi /etc/passwd`
  fornisce in output le linee del file `/etc/passwd` che contengono la stringa fissata `rossi`
- `egrep -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
- `egrep '[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.

Qualche esempio per iniziare

Andiamo sul classico:

hello.sh

```
#!/bin/bash
# Questo e' un classico (e dovuto) esempio iniziale

echo "Hello world!"

exit 0
```

I parametri

Quando si invoca uno script, così come per ogni comando da linea di comando, è possibile specificare alcuni parametri di esecuzione. Da uno script shell è possibile recuperare questi parametri utilizzando alcune variabili speciali:

- `$0`: il nome dello script che è stato invocato;
- `$1`, ..., `$9`: `$i` corrisponde all'`i`-esimo parametro passato;
- `$*`: corrisponde all'espansione `$1 $2 $3 ...`. Se usato con il double-quoting corrisponde a `"$1 $2 $3 ..."`;
- `$@`: simile al precedente, ma, se usato con il double-quoting corrisponde a `"$1" "$2" "$3" ...`.
- `$#`: il numero di parametri distinti passati

Se vengono passati più di 9 parametri, si può utilizzare il comando `shift` per accedere ai successivi: il comando effettua uno "shift" dei comandi verso sinistra "buttando via" il primo parametro (`$1`).

Strutture condizionali

Tra i costrutti disponibili, quello fondamentale è il classico if-then-else.

La sintassi è la seguente:

```
if test-commands; then
    commands;
[ elif test-commands; then
    commands; ]
[ else commands; ]
fi
```

Se l'exit status di `test-commands` è 0 allora la lista di comandi specificata in `commands` viene eseguita. Se sono presenti le clausole `elif` queste vengono valutate in sequenza con la stessa semantica.
Se nessuno dei `test-commands` ha exit status pari a 0 allora i comandi specificati nella clausola `else` (se presente) vengono eseguiti.

Esempi con if-then-else

checkstring.sh

```
#!/bin/bash
# controlla se il file passato come secondo parametro contiene
# la stringa passata come primo parametro

if grep -q "$1" "$2" ; then
   echo "Il file $2 contiene almeno almeno un'occorrenza di '$1'."
   exit 0
else
   echo "Il file $2 non contiene alcuna occorrenza di '$1'."
   exit 1
fi
```

checkwithcat.sh

```
#!/bin/bash
# controlla se esiste il file passato come parametro utilizzando
# il codice di errore riportato dal comando cat per il controllo

if cat "$1" >/dev/null 2>/dev/null
then
   echo "Il file $1 esiste ed e' leggibile."
else
   echo "Il file $1 non esiste o non hai il permesso di lettura."
   exit 1
fi
```

Espressioni condizionali

Per controllare più semplicemente certe condizioni è possibile utilizzare il comando builtin `[ ... ]` per costruire delle espressioni condizionali.
L'exit status di questo comando dipende dalla valutazione delle condizioni contenute tra le parentesi quadre.
Esistono molti operatori (unari e binari) che è possibile utilizzare. Per operare con le stringhe i principali operatori sono:

- `str1 = str2`: le stringhe sono uguali
- `str1 != str2`: le stringhe sono diverse
- `-z str`: la stringa `str` è nulla
- `-n str`: la stringa `str` non è nulla

Quando si fanno i controlli sulle stringhe è sempre bene usare il double-quoting.

Esempio con le stringhe

checkuser.sh

```
#!/bin/bash
# controlla che almeno un parametro sia stato passato e
# che sia uguale al nome dell'utente corrente

if [ -z "$1" ]; then
   echo "utilizzo: `basename $0` nome"
   exit 1
fi

CURRENT_USER=`whoami`

if [ "$CURRENT_USER" = "$1" ]; then
   echo "Il parametro passato è uguale all'utente corrente."
else
   echo "Il parametro non corrisponde all'utente corrente."
   exit 1
fi

exit 0
```

Espressioni condizionali (2)

Per i controlli sui numeri interi:

- `num1 OP num2`: l'operatore `OP` è un operatore aritmetico e può essere `-eq`, `-ne`, `-lt`, `-le`, `-gt`, `-ge` (rispettivamente: uguale, diversi, minore di, minore uguale, maggiore di, maggiore uguale)

Controlli su file:

- `-e file`: vero se il file esiste
- `-d file`: vero se il file esiste ed è una directory
- `-f file`: vero se il file esiste ed è un file regolare

Espressioni condizionali (3)

Abbiamo la possibilità di utilizzare anche alcuni connettivi logici:

- `! exp`: nega il valore booleano della condizione
- `exp1 -a exp2`: restituisce vero se sia `exp1` che `exp2` sono vere
- `exp1 -o exp2`: restituisce vero se almeno uno tra `exp1` ed `exp2` è vera

La struttura di selezione case (1)

Anche nella bash-scripting esiste il comando `case` che è l'analogo dello switch del C/C++. Permette di dirigere il flusso del programma ad uno dei diversi blocchi di codice, in base alle condizioni di verifica. È una specie di scorciatoia di enunciati if/then/else multipli.

La sintassi è la seguente:

```
case $variabile in
    $condition1 )
        commands...
    ;;
    $condition2 )
        commands...
    ;;
esac
```

Le varie condizioni (stringhe semplici o pattern) vengono valutate in sequenza finché non ne viene trovata una vera: il corrispondente blocco viene eseguito.

Il ciclo `while`

Un altro costrutto immancabile è il ciclo `while`. La sintassi è la seguente:

```
while test-commands; do commands; done
```

Finché l'exit status dei `test-commands` è 0 esegue i comandi specificati da `commands`.

Esempio con while

countdown.sh

```
#!/bin/bash
# fa un conto alla rovescia partendo da un numero passato
# da tastiera fino a zero. Se non specificato, il conteggio parte da 10

if [ ! -z "$1" ]; then
   COUNT="$1"
else
   COUNT=10
fi

echo "Inizio conto alla rovescia:"
while [ "$COUNT" -ge 0 ]; do
   sleep 1
   echo "$COUNT"
   let "COUNT = $COUNT - 1"   # decrementa COUNT
done

echo "Fine conteggio"
exit
```

Il ciclo `for`

Esiste anche una versione particolare del ciclo `for`:

`for name [in words...]; do commands; done`

Esegue i comandi specificati da `commands` per ogni stringa presente nell'espansione di `words...`. La stringa corrente viene collegata alla variabile specificata da `name`.

Esempi con `for`

params4bis.sh

```
#!/bin/bash
# legge tutte i parametri usando un ciclo 'for'

echo "Parametri:"
for P in "$@"; do
   echo "$P"
done

exit
```

Esempi con for

matrix.sh

```
#!/bin/bash
# utilizzo: matrix [rows [columns]]
# disegna una matrice di coordinate cartesiane

if [ -z "$1" ]; then
   ROWS=10
else
   ROWS=$1
fi

if [ -z "$2" ]; then
   COLUMNS=6
else
   COLUMNS=$2
fi

for R in `seq $ROWS`; do
   for C in `seq $COLUMNS`; do
      echo -n -e "($R,$C)\t"
   done
   echo
done

exit
```

Esempio: Semplice Backup

backup.sh

```
#!/bin/bash

# prendo la data e l'ora corrente
DATA=`date +%F`
ORA=`date +%H.%M`
echo "La data di oggi e' $DATA e adesso sono le ore $ORA."

DIR_BACKUP="backup_$DATA_$ORA"
echo "la directory di backup sara' $DIR_BACKUP"

# mi assicuro che la directory di backup esista
if [ ! -d "$DIR_BACKUP" ]; then
    mkdir "$DIR_BACKUP"
fi

# esamino tutti i file nella cartella corrente
for FILE in `ls -1`; do
    # controllo che si tratti di un file
    if [ -f "$FILE" ]; then
   echo "- faccio il backup di '$FILE'..."
   # ne faccio una copia compressa di backup
   cat "$FILE" | gzip > "$DIR_BACKUP/$FILE.gz"
    fi
done

echo " ...fatto!"
```