

## Introduzione

### Sistema Operativo UNIX

1970 - sviluppato nei Bell Labs di AT&T da D. Richie e K. Thompson

1976 - v6 prima versione distribuita all'esterno di AT&T

1991 - UNIX SYSVR4 e Linux 0.01

Caratteristiche attuali

- SO multiutente e multitasking
- memoria virtuale

Elementi di base

- processore comandi (interprete o shell)
- nucleo (primitive di sistema)
- linguaggio di sistema (C)

## Login/logout

- Ogni utente riceve una coppia *username* e *password*

Autenticazione per l'accesso al sistema (login):

Username: user123

Password:

- Ogni utente ha un direttorio di default (*home*) che è il direttorio corrente dopo il login
- L'uscita dal sistema va richiesta con `logout` (oppure `exit` oppure `^D`)

Lo username root è riservato all'amministratore di sistema

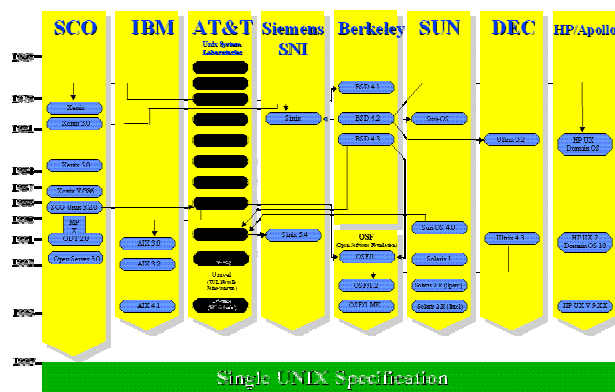
## Introduzione

### Sistema Operativo UNIX

Due linee principali

- System V (AT&T)
- BSD 4.X (Berkeley Software Distribution)

### UNIX Chronology



## File system

File System (FS) : organizza l'informazione in *file/direttori*

Due aspetti del FS di UNIX:

- omogeneità tra dispositivi e file
- i file sono stream di byte (nessuna organizzazione logica/record da parte del SO)

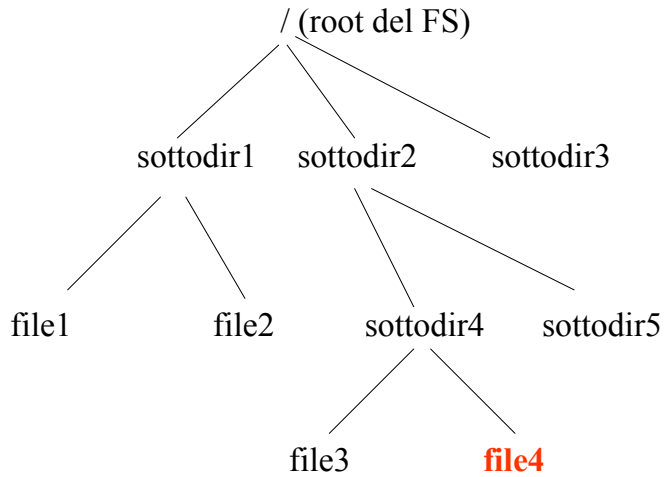
File System gerarchico: organizzazione ad albero

nodo interno = *sottodirettorio*

nodo foglia = *file*

## File system

### Organizzazione del FS



Nome assoluto: `/sottodir2/sottodir4/file4`

Nome relativo (dal direttorio corrente sottodir2): `sottodir4/file4`

## File system

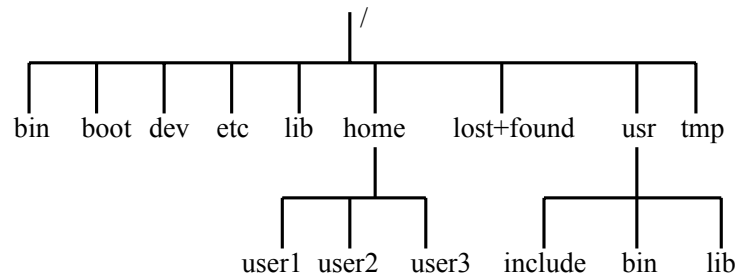
### Protezione del FS

- E' necessario regolare l'accesso alle informazioni
- Per ogni file/direttorio vengono definite tre classi di utenti
  - il proprietario (**user**)
  - il gruppo del proprietario (**group**)
  - tutti gli altri utenti (**others**)
- Per ogni tipo di utilizzatore vengono definiti tre modi di accesso:
  - lettura (**r**)
  - scrittura (**w**)
  - esecuzione (**x**) (*per i direttori regola l'accesso*)

## File system

### Struttura di un tipico FS UNIX

- molti direttori hanno un ruolo specifico



**bin** comandi principali di sistema

**dev** file speciali associati ai dispositivi

**etc** file di configurazione del sistema

**lib** librerie di sistema

**/usr/bin** altri comandi

**/usr/include** header per linguaggio C

**/home/user** home degli utenti

## File system

Ogni utente ha un identificatore (user ID - codice numerico associato al suo username) e uno o più gruppi (group ID)

- Ogni file/direttorio è associato a:
  - user-id del proprietario
  - group-id del proprietario
  - insieme di 12 bit di protezione

12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	1	1	1	0	0	1	0	0
SUID	SGID	sticky	<b>R</b>	<b>W</b>	<b>X</b>	<b>R</b>	<b>W</b>	<b>X</b>	<b>R</b>	<b>W</b>	<b>X</b>
			<b>User</b>			<b>Group</b>			<b>Others</b>		

- i primi 9 sono triple di permessi che abilitano (**r, w, x**) a ciascuna classe di utilizzatore (**U, G, O**)

## File system

12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	1	1	1	0	0	1	0	0
SUID	SGID	sticky	R	W	X	R	W	X	R	W	X
			User			Group			Others		

Il dodicesimo bit è detto set-user-id-bit

- Se è a 1 l'user-ID effettivo dell'utente diventa uguale a quello del proprietario del file per la durata dell'esecuzione del programma/script
- Necessario per comandi che accedono/modificano a risorse di root (ad es. `passwd`)
- Problemi di sicurezza

L'undicesimo bit è detto set-group-id-bit

- come SUID ma per il group-id

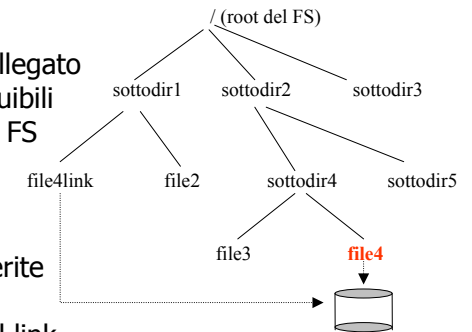
Il decimo bit è detto sticky bit (diversi significati)

## File system

### Linking

Due tipi di link

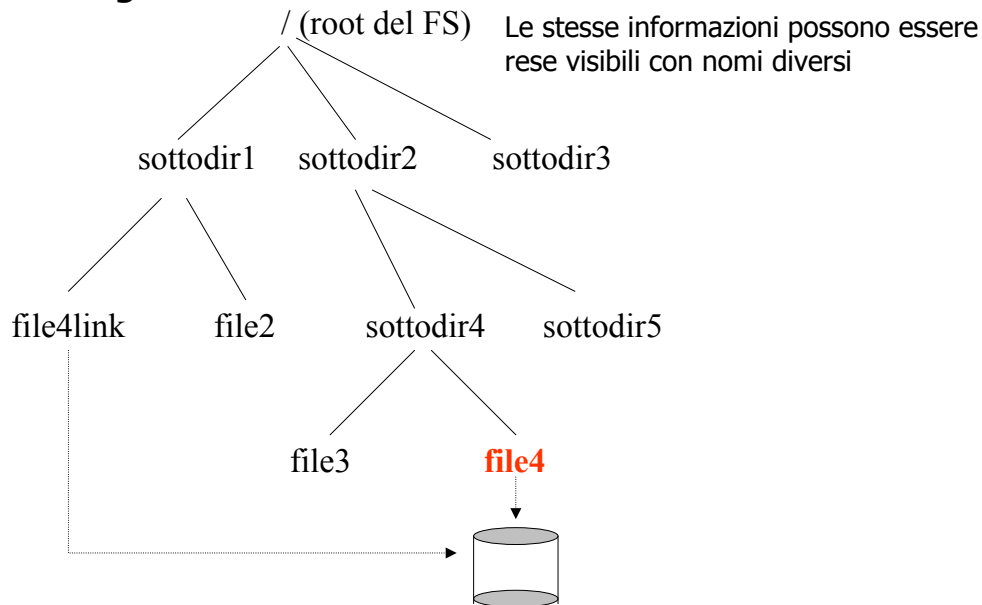
- **hard link**
  - un nuovo nome per l'oggetto collegato
  - il link e l'originale sono indistinguibili
  - condividono lo stesso i-node nel FS
  - limitazioni varie
- **symbolic link**
  - sono file speciali
  - le operazioni di I/O vengono riferite all'oggetto collegato
  - la cancellazione opera invece sul link
  - opzione `-s` per il comando `ln`



```
ln /sottodir2/sottodir4/file4 /sottodir1/file4link
```

## File system

### Linking



## Comandi

### Shell

- Lo shell mette in esecuzione i comandi forniti uno dopo l'altro (modalità interprete comandi - interattiva)

```
loop forever  
<accetta comando da console>  
<esegui comando>  
end loop;
```

- Accetta comandi anche da un file comandi fino alla fine del file (modalità processore comandi - interprete script)

```
loop forever  
<LOGIN>  
repeat  
  <accetta comando da console/file>  
  <esegui comando>  
until <fine file>  
<LOGOUT>  
end loop;
```

Vari shell disponibili

- bourne shell
- bash
- csh
- tcsh

## Comandi

### Sintassi generale

comando [-opzioni] [argomenti] <CR>

Sulla stessa linea si possono separare più comandi con ;  
(esecuzione sequenziale)

comando1 ; comando2 ...

### Comandi relativi al FS

#### Gestione direttori

mkdir <nomedir>

rmdir <nomedir>

cd <nomedir>

ls <nomedir> (lista il contenuto direttorio)

## Comandi

### Protezione nel FS

chmod [u g o] [+ -] [rwx] <nomefileodirettorio>

oppure

chmod **nuovidiritti**, <nomefileodirettorio>

- Il proprietario del file/direttorio può modificarne i diritti

#### Esempio

12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	0	1	0	0	1	0	0
SUID	SGID	sticky	R	W	X	R	W	X	R	W	X
			User			Group			Others		

chmod **ug+x** miofile

12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	1	1	0	1	1	0	0
SUID	SGID	sticky	R	W	X	R	W	X	R	W	X
			User			Group			Others		

chmod **554** miofile

## Comandi

### Trattamento file

ln <nomefile> <nomelink>

cp <filesorg> <filedest>

mv <nomefile> <nuovonomefile>

rm <nomefile>

cat <nomefileditesto> (visualizza il contenuto)

file <nomefile> (identifica il tipo di file)

### Esempi

cd /tmp

cat .cshrc

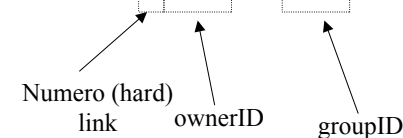
ls /bin

rm \*

## Comandi

### Altre informazioni (ottenute con ls -l)

-rwxr-xr-x	1	root	root	2612	Mar	7	2000	arch
-rwxr-xr-x	1	root	root	60592	Feb	3	2000	ash
-rwxr-xr-x	1	root	root	263064	Feb	3	2000	ash.static
-rwxr-xr-x	1	root	root	9968	Feb	3	2000	aumix-minimal
lrwxrwxrwx	1	root	root	4	Sep	22	2000	awk -> gawk
-rwxr-xr-x	1	root	root	5756	Mar	7	2000	basename



chown **nomeutente** <nomefileodirettorio>

chgrp **nomegruppo** <nomefileodirettorio>

Solo l'amministratore (root) può modificare la proprietà di file altrui

ls -a [nomedirettorio] (per visualizzare file il cui nome inizia con "." che sono normalmente nascosti)

ls -a .cshrc

## Comandi

### Comandi di stato

date  
time <nomecomando> (visualizza il tempo di esecuzione)  
who (visualizza gli utenti correnti)  
ps (visualizza i processi correnti)  
top (visualizza e ordina i processi correnti e stato del sistema)  
free (visualizza lo stato di occupazione della memoria del sistema)

### Altri comandi che operano su file di testo

more <nomefile> (visualizza il contenuto a pagine - meglio usare less)  
sort <nomefile> (ordina le righe - molte opzioni)  
diff <nomefile1> <nomefile2>  
wc [-lwc] [<nomefile>] (conta line/parole/caratteri)

## Comandi

### Manuale on-line

man -s 2 read (per ricercare solo in una certa sezione - utile se vi sono omonimie tra comandi e system call)

Per conoscere in quali direttori vengono cercate le pagine di manuale:

manpath (oppure visualizzare la var. di ambiente MANPATH)

### Altri comandi

apropos <stringa> (ricerca la presenza della stringa nel DB della descrizione dei comandi)

whatis <parola> (ricerca la presenza della parola intera nel DB della descrizione dei comandi)

## Comandi

### Manuale on-line

documenta i comandi (sez. 1), le system call (sez. 2) e altro (...)

Uso: man <nome> (ad. es. man man)

num. sezione → man(1)

sinossi → NAME

descrizione del comando → DESCRIPTION

significato delle opzioni → OPTIONS

```
man(1)
NAME
man - format and display the on-line manual pages
manpath - determine user's search path for man pages

SYNOPSIS
man [-acdfFhkkttW] [--path] [-n system] [-p string] [-C
config_file] [-M pathlist] [-P pager] [-S section_list]
[section] name ...

DESCRIPTION
man formats and displays the on-line manual pages. If you
specify section, man only looks in that section of the
manual. name is normally the name of a command, function, or
file. However, if name contains a slash (/) then man
interprets it as a file specification, so that you can do
man ./foo.5 or even man /cd/foo/bar.1.gz.

See below for a description of where man looks for the
manual page files.

OPTIONS
-C config_file
Specify the configuration file to use; the default
is /etc/man.config. (See man.conf(5).)

-M path
Specify the list of directories to search for man
pages. Separate the directories with colons. An
empty list is the same as not specifying -M at all.
See SEARCH PATH FOR MANUAL PAGES.

-P pager
Specify which pager to use. This option overrides
```

Versione HTML

in italiano su

<http://www.pluto.linux.it/ildp/man>

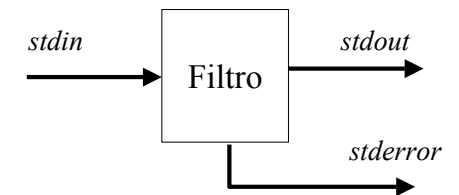
## Comandi

### Redirezione dell' I/O

Molti comandi di UNIX sono **filtri**

- possono leggere i dati di ingresso da file o dallo *standard input*
- producono risultati sullo *standard output*
- si possono combinare tra loro per ottenere comandi più complessi

Tutti i processi UNIX (non solo i filtri) dispongono dei tre canali logici di ingresso, uscita ed errore



Normalmente questi canali sono associati al terminale in uso (ad. es. console).

La redirezione permette di modificare questa associazione senza cambiare il comando:

```
ls (visualizza a schermo)
ls >listadeimieifile (l'output di ls viene rediretto sul file)
```

## Comandi

### Redirezione dell' input

```
<comando> < <fileinput>
```

### Redirezione dell' output

```
<comando> > <fileoutput>
```

```
<comando> >> <fileoutput> (output concatenato)
```

### Redirezione dell' output e di error

```
<comando> >& <fileinput>
```

### Osservazioni

- E' lo shell che riconosce la redirezione e la applica prima di eseguire il comando (vedremo più avanti come)
- si può ridirigere l'I/O sui file speciali associati ai dispositivi (/dev/printer)

## Comandi

### Alcuni filtri UNIX

(more, sort, wc)

```
grep "stringa" [nomefile] (ricerca l'occorrenza della stringa nello stdin o nel file)
```

```
tee <nomefile> (copia lo stdin in stdout ma anche nel file)
```

```
head [-numerolinee] [nomefile]
```

```
tail [-numerolinee] [nomefile]
```

```
awk [-opzioni] [nomefile] (ling. di programmazione orientato all'elaborazione di testi basato su pattern/rule)
```

### Osservazione:

Gli innumerevoli filtri UNIX possono essere utilizzati come blocchi elementari per costruire elaborazioni più complesse mediante il costrutto di **piping** di comandi

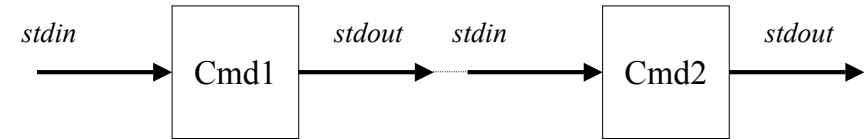
## Comandi

### Piping di comandi

#### Costrutto UNIX per il collegamento automatico di comandi

```
<comando1> | <comando2> | ... | <comandoN>
```

il *piping* collega lo stdout di un comando con lo stdin del successivo



In UNIX il piping è un costrutto parallelo: ogni comando è mappato su un processo che procede concorrentemente agli altri

In DOS il piping è implementato mediante file temporanei

```
<comando1> > filetemp ; <comando2> < filetemp
```

## Comandi

### Esempi di piping di comandi

```
ls /bin | wc -l
```

```
ps -elf | grep mionomeutente
```

```
who | awk '{print $1}' | uniq | wc -l
```

## Shell

Esecuzione di un comando in shell

Se il comando non è interno (*built-in*) viene eseguito da un nuovo shell attivato dallo shell corrente

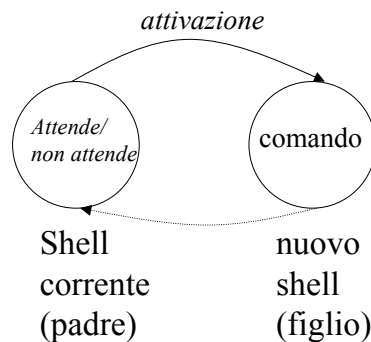
Il nuovo shell effettua nell'ordine:

1) le sostituzioni nella linea di comando

- Variabili d'ambiente
- Sostituzione dei comandi
- Metacaratteri

2) la ricerca del comando

3) l'esecuzione del comando



## Shell

Variabili di shell

Ogni shell mantiene un insieme di variabili che ne modificano il funzionamento:

- le variabili interne sono private a ciascun shell
- le variabili d'ambiente (*environment*) sono rese disponibili (copiate) ai processi figli

Le variabili hanno un nome ed un valore (stringa) :

- i riferimenti ai valori si esprimono con  $\$nomevariabile$
- la sintassi di assegnamento dipende dal tipo di shell
  - `setenv X pippo` (tcsh - variabile d' ambiente)
  - `set X=pippo` (tcsh - variabile interna)
  - `Y=$X` (bourne/bash - variabile interna)
  - `export Y` (bourne/bash - inserita nell'ambiente)

## Shell

Due modalità di esecuzione dei comandi

- *Foreground*

lo shell padre **attende** il completamento dell'esecuzione del comando (default)

- *Background*

lo shell padre **non attende** il completamento dell'esecuzione del comando (& alla fine della linea)

```
bash$ ls -lR >mieifile &
[1] 23486
bash$
bash$ date
Thu Mar 14 11:38:27 CET 2002
bash$
[1]+ Done ls -lR >mieifile
bash$
```

La redirectione (soprattutto dell'input) è necessaria  
identificatore del nuovo processo

Lo shell padre è immediatamente disponibile

Lo shell padre è informato del completamento del comando

## Shell

Esempi di variabili di shell

- variabili d'ambiente
    - PATH indica i direttori in cui ricercare i comandi
    - SHELL indica il tipo di shell di default dell'utente
    - HOME indica il direttorio di accesso dell'utente (~ equivale a \$HOME per bash/tcsh)
  - variabili interne
    - prompt (tcsh - configura la stringa di prompt dello shell)
    - status (tcsh - contiene il valore di uscita dell'ultimo comando)
- Sono assegnate in fase di login*

Per visualizzare tutte le variabili `printenv` (per var. ambiente) e `set` (per var. interne)

## Shell

### Sostituzione comandi

```
set mieifileC=`ls *.c`
```

i comandi compresi tra i backquote vengono eseguiti e viene prodotto il risultato

### Metacaratteri

Molti caratteri hanno un significato speciale nella linea di comando (alcuni già visti: > < | & \$ ` )

- # : commento (la linea non viene eseguita)
- ! : accede al meccanismo di storia dei comandi (*history*)
  - !! (ultimo comando eseguito)
  - !abc (ultimo comando che inizia con abc)
- \* ? e altri : pattern-matching con i nomi dei file

## Shell

### Controllo sulla espansione della linea di comando

Lo shell esegue di norma le seguenti sostituzioni

- 1) Variabili d'ambiente
- 2) Sostituzione dei comandi
- 3) Metacaratteri

' (quote) non permette alcuna espansione (nessuna sostituzione - né 1 né 2 né 3)

" (double quote) permette le sole sostituzioni 1 e 2 (non la 3)

Esempio

```
set y=3
```

```
echo '* e $y' # produce * e $y
```

```
echo "*" e $y" # produce * e 3
```

## Shell

### Metacaratteri

- \* : una qualunque stringa di zero o più caratteri in un nome di file/direttorio
- ? : un qualunque carattere in un nome di file/direttorio
- [c<sub>1</sub>c<sub>2</sub>...c<sub>n</sub>] o [c<sub>1</sub>,c<sub>2</sub>..., c<sub>n</sub>] : un qualunque carattere in un nome di file/direttorio incluso in quell'insieme
- [c<sub>1</sub>-c<sub>n</sub>] : un qualunque carattere in un nome di file/direttorio compreso nell'intervallo indicato

### Esempi

```
ls [ab]*.c
```

```
ls file[0-9].?
```

```
ls *\?* Il carattere \ (backslash) priva un metacarattere del suo significato
```

## Shell

### Programmazione nello shell

Gli shell UNIX sono processori di comandi:

- interpreti del proprio linguaggio comandi (sintassi) simile ad un normale linguaggio di programmazione :
  - istruzioni per il controllo di flusso (*if/case/for/while/...*)
  - variabili (*bash/tcsh* anche variabili numeriche)
  - passaggio dei parametri
  - funzioni (*sh/bash*)
- consentono la rapida prototipazione di applicazioni (in alternativa al C o agli interpreti *perl/python/...*)

I file comandi sono generalmente detti **script**

## Shell

### Esecuzione di uno script

Due possibilità

- rendere eseguibile lo script e lanciarlo in esecuzione:

```
chmod +x mioscript ; mioscript (viene messo  
automaticamente in esecuzione uno shell)
```

- invocare uno shell per eseguirlo:

```
sh mioscript (l'opzione -x mostra l'esecuzione di ciascun  
comando)
```

E' bene esplicitare nel file comandi l'interprete richiesto per l'esecuzione inserendo un commento speciale all'inizio del file:

```
#!/bin/tcsh  
echo Script running  
...
```

In assenza del commento UNIX mette in esecuzione sh

## Shell

### Altre variabili

- \$\* l'insieme di tutte le variabili posizionali (tutti gli argomenti)
- \$# il numero di argomenti di attivazione (\$0 escluso)
- \$\$ l' identificatore del processo in esecuzione (PID)
- \$? lo stato (valore di uscita) dell'ultimo comando eseguito

L' esecuzione di un comando produce un valore di uscita (anche in \$status) che può essere resa parte di una espressione (istruzioni per il controllo di flusso) :

valore zero ⇒ esecuzione riuscita  
valore positivo ⇒ errore

### Esempio

```
cp miofile $DIR ; echo $status (0 ⇒ OK ; >0 ⇒ è fallito (esistenza  
del file/diritti/spazio nel FS/...)  
  
grep stringa fileditesto (0 ⇒ OK il file contiene la stringa)
```

## Shell

### Passaggio dei parametri

Gli argomenti di invocazione dello script sono disponibili in variabili posizionali:

```
mioscript argomento1 argomento2 ... argomentoN  
variabile $0 : il comando  
variabile $1 : il primo argomento  
variabile $2 : il secondo argomento  
...
```

### Esempio

lo script DIR1 contiene `ls ~/$1`  
invocato con DIR1 bin: \$0 vale DIR1  
\$1 vale bin

## Shell

### Alcuni comandi e istruzioni per script Bourne/bash

- **test** -opzioni condizione : comando per valutare varie condizioni (espressioni e condizioni sui file -f -d -r )
- **if ( <comandi> )** : istruzione condizionale  
**then <comandi> [else <comandi> ] fi**
- **for <var> [in <list> ] do <comandi> done** : istruzione per ripetizione enumerativa
- **while <comandi> do <comandi> done** : istruzione per ripetizione non enumerativa
- **case <var> in <pattern-1> <comandi> ... esac** : istruzione per alternativa multipla
- **read <var>** : comando per l'input di una variabile da stdin
- **echo <stringa>** : comando per visualizzare stringhe (echo \$newvar)
- **exit [status]** : istruzione per la terminazione dello script (con eventuale valore di uscita)

## **Shell**

Un semplice esempio di script per spostare in un direttorio tutti i file di una certa estensione che contengono una certa parola:

### Invocazione

```
sposta estensione parola direttorio
#!/bin/sh
if (test $# -ne 3) then
    echo "Uso: $0 estensione parola direttorio"
    exit -1
fi
if (test ! -d $3) then
    echo "Il direttorio $3 non esiste"
    exit -2
fi
for i in *.$1
do
    echo "Esamino il file $i"
    if (grep $2 $i ) then
        echo "Sposto $i in $3"
        mv $i $3
    fi
done
```