

I Comandi di Unix

Classi di comandi Unix

- Gestione dei file e delle directory
- Sviluppo software
- Elaborazione testi
- Amministrazione del sistema
- Comunicazione
- ...

Formato dei comandi

comando [argomento ...]

Gli argomenti possono essere:

- **opzioni o flag** (-)
- **parametri**

separati da almeno un separatore

Esempio

ls **-l -F** file1 file2 file3

Forme equivalenti:

ls -F -l file1 file2 file3

ls -lF file1 file2 file3

ls -lF file1 file2 file3

Il comando `man`

Unix ha un manuale di riferimento, accessibile “in linea”, mediante il comando **`man`**

Il manuale è organizzato in **sezioni** e **sottosezioni**; ogni sezione è composta di **pagine** (logiche)

Ogni pagina descrive **un singolo argomento** (es.: un comando)

Le sezioni del manuale

Sezione	Contenuti
1	Commands
2	System Calls
3	Library Functions
4	Administrative Files
5	Miscellaneous Information
6	Games
7	I/O and Special Files
8	Maintenance Commands

Il comando **man**

man [opzione...] titolo...

Visualizza le pagine del manuale specificate mediante i suoi parametri

-s permette di specificare la sezione

Esempio

```
%man who
```

```
....
```

```
%man -s 2 kill
```

```
....
```

```
%man -s 2 intro
```

Note:

- Se il numero di sezione non è specificato, viene selezionata la prima occorrenza
- Ogni sezione o sottosezione inizia con una pagina

chiamata **intro**

% man man

man(1)

man(1)

NAME

man - format and display the on-line manual pages

manpath - determine user's search path for man pages

SYNOPSIS

man [-acdfFhkKtwW] [-m system] [-p string] [-C config_file] [-M path] [-P pager] [-S section_list] [section] name ...

DESCRIPTION

man formats and displays the on-line manual pages. This version knows about the MANPATH and (MAN)PAGER environment variables, so you can have your own set(s) of personal man pages and choose whatever program you like to display the formatted pages.

<omissis>

OPTIONS

-C config_file

Specify the man.conf file to use; the default is /etc/man.config. (See man.conf(5).)

<omissis>

OPERANDS

The following operand is supported:

name A keyword or the name of a standard utility.

USAGE

Manual Page Sections

Entries in the reference manuals are organized into sections. A section name consists of a major section name, typically a single digit, optionally followed by a subsection name, typically one or more letters.

<omissis>

ENVIRONMENT

See `environ(5)` for descriptions of the following environment

<omissis>

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

FILES

`/usr/share/man` root of the standard manual
page directory subtree

<omissis>

SEE ALSO

`apropos(1)`, `cat(1)`, `col(1)`, `eqn(1)`, `more(1)`, `nroff(1)`,
`refer(1)`, `tbl(1)`, `troff(1)`, `vgrind(1)`, `whatis(1)`,
`catman(1M)`, `attributes(5)`, `environ(5)`, `eqnchar(5)`, `man(5)`

Gestione delle Directory

Gestione delle directory

pwd	p rint w orking d irectory
cd	c hange d irectory
ls	l ist directory
du	d isk u sage
mkdir	m ake d irectory
rmdir	r emove d irectory
ln	l ink
...	

Il comando `pwd`

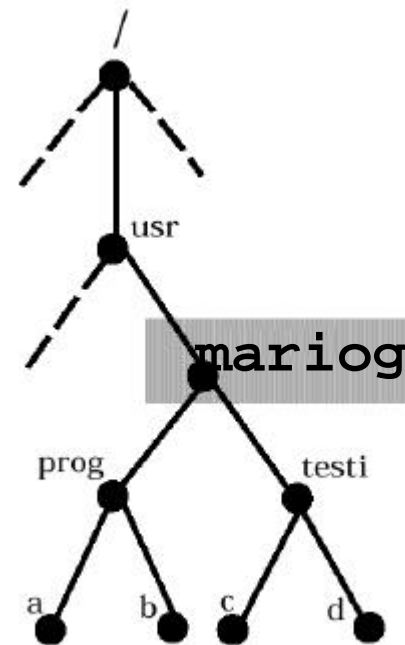
`pwd`

"print working directory"

stampa il path della directory corrente

Esempio:

```
% pwd
/usr/mariog
%
```



Il comando cd

cd [directory]

"change directory"

- la directory specificata diviene la **working directory**
- se nessuna directory è specificata, si "ritorna" alla home directory

Esempio

```
%cd /usr
```

```
%pwd
```

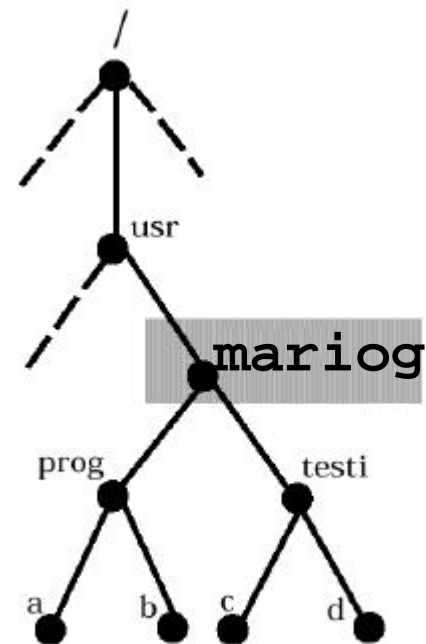
```
/usr
```

```
%cd
```

```
%pwd
```

```
/usr/mariog
```

```
%
```



Il comando ls

ls [options][directory...]

"list directory"

- lista (in ordine alfabetico) il contenuto della o delle directory indicate
- se nessuna directory è indicata, lista il contenuto della working directory
- possiede numerose opzioni

Il comando ls: alcune opzioni

- s fornisce la dimensione in blocchi (size)
- t lista nell'ordine di modifica (prima il file modificato per ultimo) (time)
- l un nome per ogni riga
- F aggiunge / al nome delle directory e * al nome dei file eseguibili
- R si chiama ricorsivamente su ogni sottodirectory
- i fornisce l'i-number del file
- e molte altre!

Esempio

```
% ls
dir1 file1
% ls -s
total 4 2 dir1 2 file1
% ls -t
file1 dir1
% ls -1
dir1
file1
% ls -F
dir1/ file1
% ls -R
dir1 file1
./dir1:
file1 file2 file3 file4
```

I file nascosti

I file il cui nome inizia con "." vengono listati solo specificando l'opzione **-a** ("all")

Esempio

```
% ls -a
. .cshrc .mailrc dir1
.. .login .sh_history file1
%
```

Il comando du

du [options][name...]

"disk usage"

- stampa il numero di blocchi contenuti in tutti i file e (ricorsivamente) directory specificate
- se **name** non è specificato, si intende la directory corrente
- **-s**: solo il totale

```
% du
2 ./dir1
2 ./dir2
14 .
% du -s ..
198812 ..
%
```

Esempio

Il comando `mkdir`

`mkdir directory...`

"make directory"

- crea la/le directory indicata/e

Esempio

```
% mkdir dir1 dir2
% ls
dir1 dir2
%
```

Il comando **rmdir**

rmdir directory

"remove directory"

- rimuove la/le directory indicata/e
- le directory devono essere vuote

Esempio:

```
% rmdir dir  
rmdir: dir: Directory not empty  
% ls dir  
a  
% rm dir/a  
  
% rmdir dir  
  
%
```

Il comando ln

In name1 name2

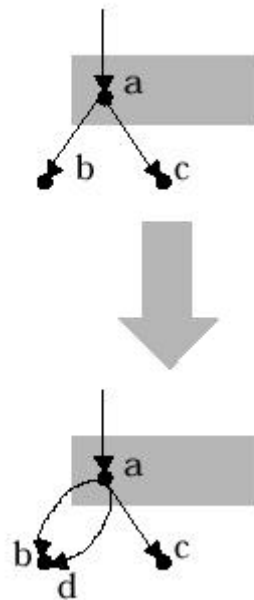
"link"

- associa il nuovo nome (link) **name2** al file (esistente) **name1**, che non può essere una directory

Esempio

```
% ln b d
```

```
%
```



Il comando `ln` (cont.)

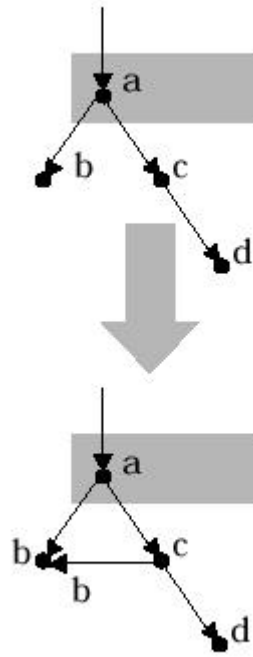
`$ ln name1 name2`

- se `name2` è una directory, il nuovo nome è `name2/name1`

Esempio

```
% ln b c
```

```
%
```

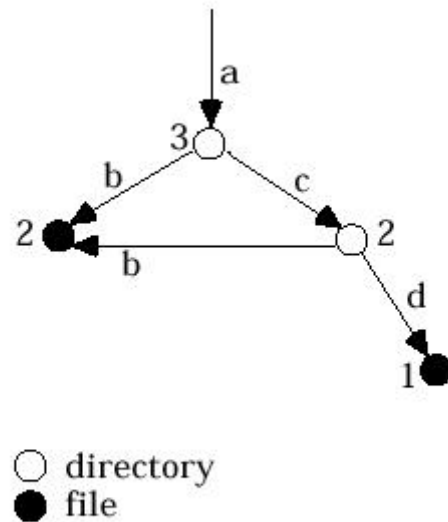


Numero di link

È uno degli attributi dei file gestiti dal sistema

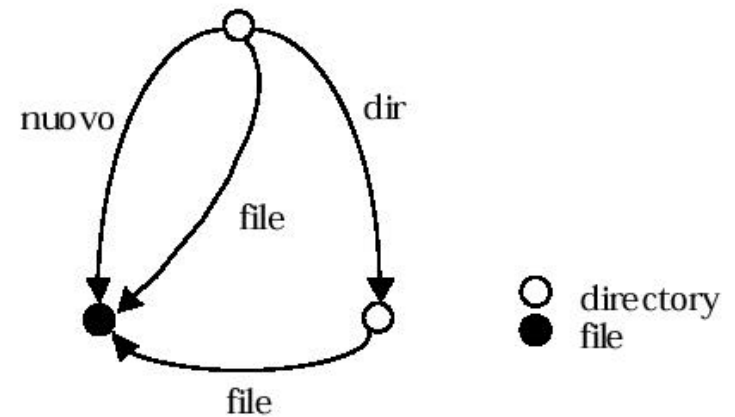
È il numero di volte che si fa riferimento all'i-number

Esempio



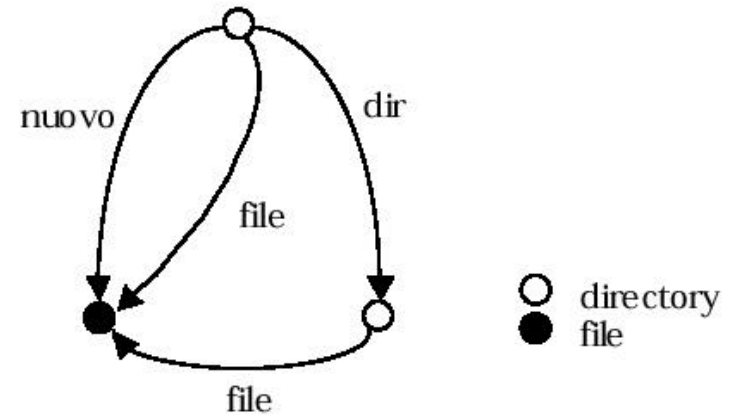
Per vedere il numero di link: **ls -l**

Esempio



```
% mkdir dir
% touch file
% ls -l
total 2
drwxr-sr-x 2 mariog staff 512 Mar 11 19:40 dir
-rw-r--r-- 1 mariog staff  0 Mar 11 19:40 file
% ln file nuovo
% ls -li
199742 dir 51204 file 51204 nuovo
```

Esempio



```
% ls -l
total 2
drwxr-sr-x 2 mariog staff 512 Mar 11 19:40 dir
-rw-r--r-- 2 mariog staff 0 Mar 11 19:40 file
-rw-r--r-- 2 mariog staff 0 Mar 11 19:40 nuovo
% In file dir
% ls -l dir
total 0
-rw-r--r-- 3 mariog staff 0 Mar 11 19:40 file
% In dir nuovissimo
In: dir is a directory
%
```

Note

- Tutti i **link** allo **stesso file** hanno identico status e caratteristiche
- Non è possibile distinguere la entry originaria dai nuovi link
- I **link** di questo tipo **non** possono essere fatti con file che si trovano su **filesystem diversi**

Link simbolici

In -s name1 name2

- Permette di creare link a directory e link fra file o directory che stanno su file system diversi
- Viene creato un file **name2** che contiene il link simbolico (path di **name1**)

Esempio

```
% ls
```

```
dir
```

```
% ls dir
```

```
file1 file2
```

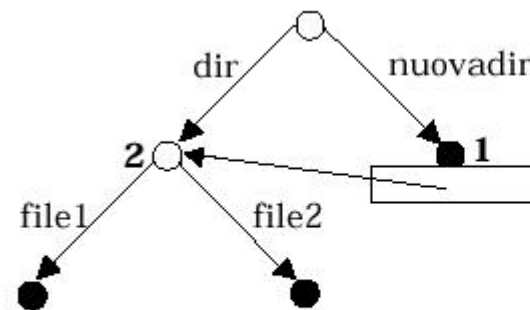
```
% ln -s dir nuovadir
```

```
% ls
```

```
dir nuovadir
```

```
% ls nuovadir
```

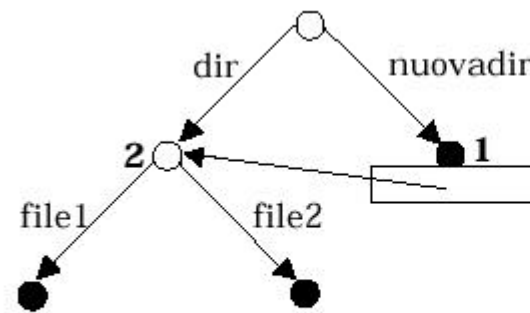
```
file1 file2
```



○ directory
● file

Esempio

```
% ls -l
total 4
drwxr-sr-x 2 mariog  staff  512 Mar 11 19:24
dir
lrwxrwxrwx 1 mariog  staff   3 Mar 11 19:24
nuovadir -> dir
%
```



○ directory
● file

Gestione dei File

Comandi di gestione dei file

mv **m**ove file

cp **c**opy file

rm **r**emove file

touch modifica data e ora dell'ultimo
accesso/modifica di un file

find cerca file con specificati attributi

...

Il comando **mv**

mv [options] name...target

"move"

- **muove** il file o directory **name** sotto la directory **target**
- se **name** e **target** non sono directory, il **contenuto** di **target** viene **sostituito** dal contenuto di **name**

Esempio

```
% ls
file1 file2 targetdir
% mv file1 file2 targetdir
% ls
targetdir
% ls targetdir
file1 file2
% mv targetdir/file1 targetdir/file2 .
% ls
file1 file2 targetdir
```

*targetdir è
una directory!*

Esempio

```
% ls
file1 file2 file3 targetfile

% mv file1 targetfile

% ls
file2 file3 targetfile

% mv file2 file3 targetfile

mv: Target targetfile must be a directory

Usage: mv [-f] [-i] f1 f2
mv [-f] [-i] f1 ... fn d1
mv [-f] [-i] d1 d2
```

*targetdir è
un file!*

Esempio

```
% ls  
file1 file2  
% mv file1 file2 target  
mv: target not found  
% mv file1 target  
% cat target  
contenuto di file1  
%
```

*targetdir
non esiste!*

Il comando cp

cp [options][name...] target

"copy"

come **mv**, ma **name** viene
copiato

Esempio

```
% ls
file1 file2 targetdir
% cp file1 file2 targetdir
% ls . targetdir
.:
file1 file2 targetdir
targetdir:
file1 file2
% ls
file1 targetfile
% cp file1 targetfile
% ls
file1 targetfile
%
```

Il comando **rm**

rm [-r] name

"remove"

- **rimuove** i file indicati
- se un file indicato è una directory:

messaggio di errore, a meno che non sia specificata l'opzione **-r**

- ... nel qual caso, rimuove ricorsivamente il contenuto della directory
- <altri parametri>

Il comando touch

touch [options][time] filename...

- aggiorna la data e l'ora dell'ultimo accesso (opzione **-a**) o dell'ultima modifica (opzione **-m**) di filename (default: **-am**)
- se **time** non è specificato, usa la data e l'ora corrente
- se il file non esiste, lo crea

Esempio:

```
% touch 01281738 file1
% ls -l
total 0
----r--r-- 1 mariog  staff 0 Jan 28 17:38 file1
```

Generazione dei nomi dei file

La shell fornisce un meccanismo che permette di specificare una lista di nomi di file mediante una singola espressione sintetica

Esempio:

```
% ls  
gianni giorgio laura mario  
  
% rm g*  
  
% ls  
laura mario  
  
% rm *  
  
%
```

*Rimuove tutti i file presenti nella directory corrente, ad eccezione di quelli che iniziano con il "."
(.profile,.cshsc,.login,...)*

Metacaratteri

? un carattere qualsiasi

Es. **rm file?**

***** una stringa di \emptyset o più caratteri qualsiasi

Es. **rm file***

[...] uno qualsiasi dei (singoli) caratteri racchiusi fra **[]** (alternativa)

Es. **rm file[123]**

rm file[a-z] (subrange)

N.B.

- il "." iniziale deve essere sempre specificato
- per usare il carattere del metacaratteri, occorre anteporre \

Il comando find

find pathname ... [expression]

discende ricorsivamente le directory specificate (**pathname...**), cercando tutti i file che rendono vera **expression**.

Molto flessibile:

- **ricerca** file di specificati **attributi** (nome, tipo, permessi, proprietario, gruppo, numero di link, dimensione, data di ultima modifica/accesso ...)
- **and, or, not** di attributi
- può **eseguire** automaticamente, o previa conferma, uno o più **comandi** sui file individuati

Esempio

Rimuovi tutti i file nella working directory (o più sotto) di nome **a.out** il cui ultimo accesso è anteriore a 7 giorni:

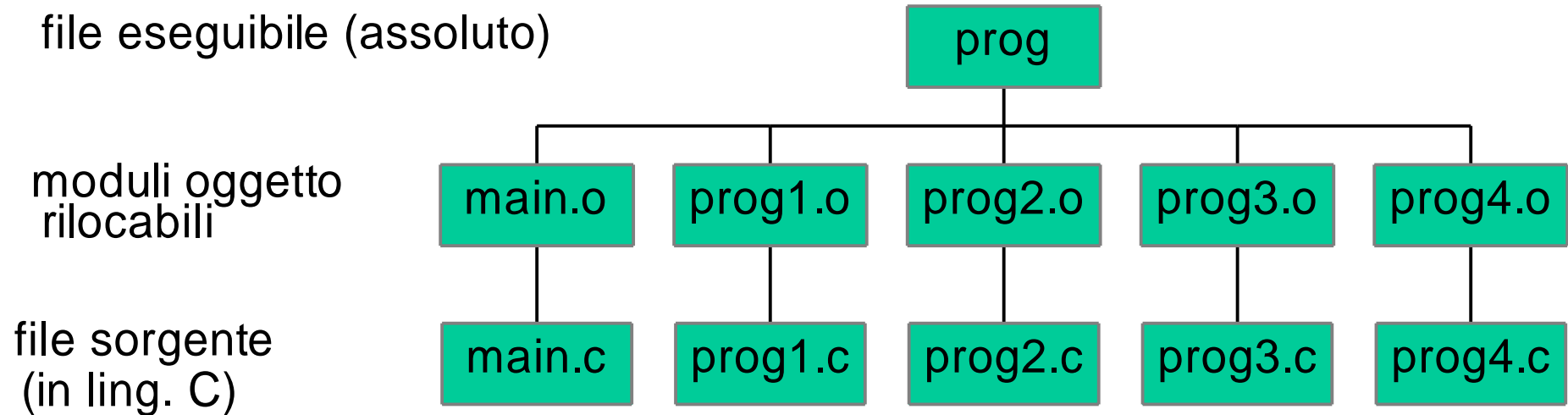
```
find . -name a.out -atime +7 -exec rm {} \;
```

Come si interpreta:

- .** directory da cui iniziare la ricerca
- name a.out** nome del file da cercare
- atime +7** data di accesso superiore a 7 giorni
- exec rm {} \;** esegui il comando **rm** sul file trovato (**\:** carattere di terminazione)

Sviluppo software

Un progetto



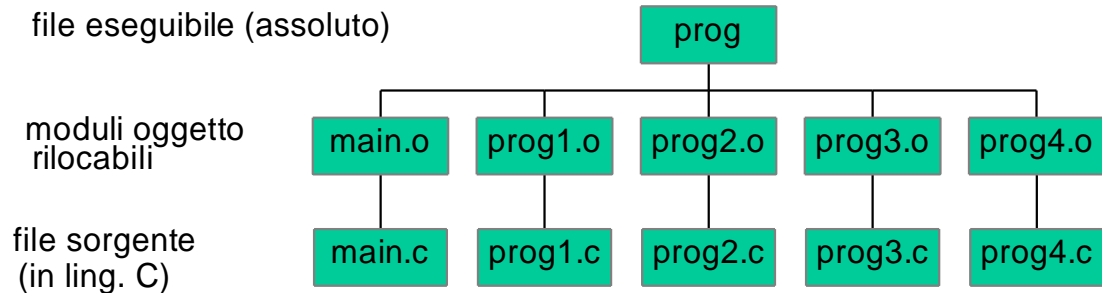
Un progetto (segue)

Supponiamo che:

- **main.o** non esista

- **prog1.o** e **prog4.o** siano antecedenti all'ultima versione di **prog1.c** e **prog4.c**

- **prog2.o** e **prog3.o** siano relativi all'ultima versione di **prog2.c** e **prog3.c**

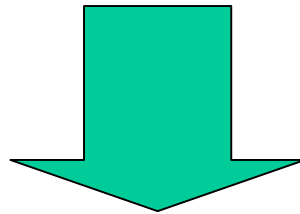


```
$ cc -c main.c
$ cc -c prog1.c
$ cc -c prog4.c
$ cc -o prog main.o prog1.o ... prog4.o
```

Un progetto (segue)

Se i sorgenti sono frequentemente modificati

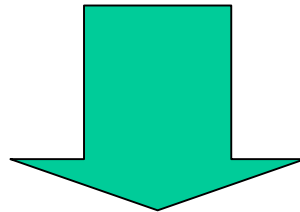
- controllare se qualche file sorgente e` stato modificato
- ricompilare gli eventuali file modificati
- ricostruire l'eseguibile prog



operazione ripetitiva

Il comando **make**

Il comando **make** consente la manutenzione e l'aggiornamento di programmi



- controlla se i file sorgente sono stati ricompilati dopo l'ultima modifica
- compila i file modificati dopo i relativi file oggetto
- ricostruisce la versione aggiornata di prog

Il comando make (segue)

Per svolgere le sue funzioni il comando **make** ha bisogno di un file ausiliario, il **makefile**

Esempio

```
# makefile per prog
# versione 1
prog: main.o prog1.o prog2.o prog3.o prog4.o
    cc -o prog main.o prog1.o ... prog4.o
main.o: main.c
    cc -c main.c
prog1.o: prog1.c
    cc -c prog1.c
...
prog4.o: prog4.c
    cc -c prog4.c
```

Linee di dipendenza

Linee di comando

Il comando make (segue)

Una **linea di dipendenza** definisce la **relazione** tra un **file** dipendente o target (a sx del carattere :) e uno o più **file di dipendenza**

Una **linea di comando** o regola, definisce le operazioni che **make** deve **effettuare** per passare da un **file di dipendenza** ad uno **dipendente**

Una linea di comando del **makefile** e` **eseguita** se i relativi **file** di dipendenza sono stati **modificati** più **recentemente** dei file dipendenti associati

Il comando make (segue)

Digitando **make** si ottiene l'esecuzione delle linee di comando relative al primo file dipendente

Se i file di dipendenza sono *superati* (*non esistono*), vengono *eseguite* anche le linee di *comando necessarie* al loro aggiornamento (*loro creazione*)

Il comando make (segue)

Non sempre e` necessario specificare tutte le dipendenze e le regole

make e` in grado di riconoscere le relazioni esistenti tra alcuni file e di applicare ad essi regole predefinite

make riconosce file con il suffisso .c come sorgenti in linguaggio C e fa eseguire, *se* necessario, la loro **compilazione**

```
# makefile di prog
# versione 2
prog: main.o prog1.o ... prog4.o
    cc -o prog main.o ...
main.o: main.c
.
.
prog4.o: prog4.c
```

Il comando make (segue)

Se uno dei file oggetto specificati e` superato, **make** ricerca nella directory corrente un file sorgente con lo stesso nome e fa eseguire la compilazione

Esempio

```
# makefile di prog
#versione 3

prog: main.o prog1.o ... prog4.o
    cc -o prog main.o ...
```

I makefile parametrici

E` possibile rendere **parametrici** i makefile utilizzando le *macro*

```
# makefile di prog
# versione 4

OGGETTI: main.o prog1.o ...
Prog: $(OGGETTI)

cc -o prog $(OGGETTI)
```

Le **macro** si usano per definire elenchi di file, opzioni dei compilatori, librerie, comandi, ...

```
csh> cat makeprova
OGGETTI = qdrig.o pippo.o
OGGETTI1 = qdcol.o pluto.o
MAIN = driver.o
MAIN1 = driver_uno.o
rows:      $(OGGETTI) $(MAIN)
           cc -o rows $(OGGETTI) $(MAIN)
columns:   $(OGGETTI1) $(MAIN1)
           cc -o columns $(OGGETTI1) $(MAIN1)
csh>make -f makeprova rows
cc -c qdrig.c
cc -c pippo.c
cc -c driver.c
cc -o rows driver.o qdrig.o pippo.o

csh>make -f makeprova rows
'rows' is up to date.
```

Esempio

```
csh>make -f makeprova -n  
cc -c qdrig.c  
cc -c pippo.c  
cc -c driver.c  
cc -o rows driver.o qdrig.o  
pippo.o  
csh> touch *.c
```

Visualizza quello che farebbe, ma non esegue niente!

Aggiorna la data di tutti i file .c

Sommario

Abbiamo visto alcuni comandi di Unix

Il comportamento dei comandi dipende dall'implementazione

Tenere sempre il manuale a portata di mano!