Fondamenti di Informatica (Elettronici)

Comandi di shell – Lezione 28

11 dicembre 2020

Comandi della Bourne shell [1]

1. comandi 2/2
2. Creazione di comandi personalizzati 1/3
3. Bash: sintassi dei comandi
4. Bash: programmazione

[1] Tratto da: Cameron Wilson, Top 25 commands and creating custom commands, 27 agosto 2019. Tratto da: Educative.blog() for developers. By developers.

Section 1

comandi 2/2

`head` - Legge l'inizio di un file

Per impostazione predefinita, il comando visualizza le prime 10 righe di un file.

Ci sono momenti in cui potresti aver bisogno di guardare velocemente alcune righe in un file e `head` ti permette di farlo.

Un tipico esempio di quando si desidera utilizzare `head` è quando è necessario `analizzare i log` o i `file di testo` che cambiano frequentemente.

Sintassi: `head [option(s)] file(s)`

Opzioni comuni: `-n`

`tail` - Legge la fine di un file

Per impostazione predefinita, il comando `tail` visualizza le ultime 10 righe di un file.

Ci sono momenti in cui potresti aver bisogno di guardare velocemente alcune righe in un file e `tail` ti permette di farlo.

Un tipico esempio di quando si desidera utilizzare `tail` è quando è necessario `analizzare i log` o i file di testo che cambiano frequentemente.

Sintassi: `tail [option(s)] file_names`

Opzioni comuni: `-n`

`chmod` - Imposta il flag dei permessi su un file o cartella

Ci sono situazioni in cui ti imbatterai in cui tu o un collega proverete a caricare un file o a modificare un documento e riceverete un errore perché non avete accesso. La soluzione rapida per questo è usare `chmod`.

Le autorizzazioni possono essere impostate con caratteri alfanumerici (`u`, `g`, `o`) e possono essere assegnate al loro accesso con w, r, x.

Al contrario, puoi anche usare numeri ottali (0-7) per modificare i permessi. Ad esempio, `chmod 777 my_file` darà accesso a tutti.

`chmod`: Quick Referance with Examples

Sintassi: `chmod [option(s)] permissions file_name`

Opzioni comuni: `-f`, `-v`

`exit` - Esce da una directory

Il comando `exit` chiuderà una finestra di terminale, terminerà l'esecuzione di uno script di shell o ti disconnetterà da una sessione di accesso remoto SSH.

Sintassi: `exit`

Opzioni comuni: `n/a`

`history` - elenca i comandi più recenti

Un comando importante quando è necessario identificare rapidamente i comandi precedenti che hai utilizzato.

Sintassi: `history`

Opzioni comuni: `-c, -d`

`clear` - Cancella la finestra del terminale

Questo comando viene utilizzato per cancellare tutti i comandi precedenti e l'output dalle console e dalle finestre del terminale.

Ciò mantiene il tuo terminale pulito e rimuove il disordine in modo che tu possa concentrarti sui comandi successivi e sul loro output.

Sintassi: `clear`

Opzioni comuni: `n/a`

cp - copia file e directory

Utilizzare questo comando quando è necessario eseguire il backup dei file.

Sintassi: `cp [option(s)] current_name new_name`

Opzioni comuni: `-r, -i, -b`

`kill` - termina i processi bloccati

Il comando `kill` consente di terminare un processo dalla riga di comando.

A tale scopo, fornire l'ID di processo (__PID__) del processo da terminare.

Per trovare il PID, puoi usare il comando ps accompagnato dalle opzioni -aux.

Sintassi: `kill [signal or option(s)] PID(s)`

Opzioni comuni: `-p`

`sleep` - ritarda un processo per un periodo di tempo dato

`sleep` è un comando comune per il controllo dei lavori ed è utilizzato principalmente negli script della shell.

Noterai nella sintassi che c'è un suffisso; il suffisso viene utilizzato per specificare l'unità di tempo se è `s` (secondi), `m` (minuti) o `d` (giorni).

L'unità di tempo predefinita è i secondi, a meno che non sia specificato.

Sintassi: `sleep number [suffix]`

Opzioni comuni: `n/a`

Section 2

Creazione di comandi personalizzati 1/3

Come creare i tuoi comandi Bash personalizzati

I comandi personalizzati in Bash sono noti come "alias".

Gli alias sono essenzialmente un'abbreviazione o un mezzo per evitare di digitare una lunga sequenza di comandi.

Possono risparmiare una grande quantità di digitazione sulla riga di comando in modo da evitare di dover ricordare complesse combinazioni di comandi e opzioni.

C'è un avvertimento nell'usare gli alias, ed è essere sicuri di non sovrascrivere nessuna parola chiave.

Sintassi: `alias alias_name='ls -l'`

Come creare i tuoi comandi Bash personalizzati 2/3

Un esempio molto semplice sarebbe simile a questo:

```
alias c='clear'
```

Ora ogni volta che vuoi cancellare lo schermo, invece di digitare clear, puoi semplicemente digitare c e sarai a posto.

Puoi anche diventare più complicato, ad esempio se volessi configurare un server web in una cartella:

```
alias www='python -m SimpleHTTPServer 8000'
```

Come creare i tuoi comandi Bash personalizzati 3/3

Ecco un esempio di un utile alias per quando devi testare un sito web in diversi browser web:

```
alias ff4='/opt/firefox/firefox'
alias ff13='/opt/firefox13/firefox'
alias chrome='/opt/google/chrome/chrome'
```

Oltre a creare alias che fanno uso di un comando, puoi anche utilizzare alias per eseguire più comandi come:

```
alias name_goes_here='activator && clean && compile && run'
```

Sebbene sia possibile utilizzare alias per eseguire più comandi, si consiglia di utilizzare le funzioni (julia) in quanto sono notevolmente più flessibili e consentono di eseguire una logica più complessa e sono ottime per la scrittura di script.

parametri, variabili, espansione e sostituzione

Un compito molto importante delle shell Unix è quello di rimpiazzare variabili e simboli speciali con quello che rappresentano.

A fianco di questo problema si pone la necessità di proteggere ciò che si vuole evitare sia espanso dalla shell.

Anche questi simboli che proteggono contro l'espansione sono soggetti a loro volta a un procedimento di sostituzione: quando la shell ha terminato l'interpretazione di un'istruzione, questi devono essere rimossi in modo da non lasciarne traccia per un eventuale programma che dovesse ricevere questi dati in forma di argomenti.

Quoting: protezione

Il quoting è un' azione con la quale si toglie il significato speciale che può avere qualcosa per la shell.

Si distinguono tre possibilità: il carattere di escape (\) (reverse slash), gli apici semplici (') e gli apici doppi (") .

In generale, il concetto può essere trasferito in quello della protezione da un'interpretazione errata di ciò che si intende veramente.

Escape e continuazione

La barra obliqua inversa (\) rappresenta il carattere di escape.

Serve per preservare il significato letterale del carattere successivo, cioè evitare che venga interpretato diversamente da quello che è veramente.

Un caso particolare si ha quando il simbolo \ è esattamente l'ultimo carattere della riga, o meglio, quando questo è seguito immediatamente dal carattere di interruzione di riga (`\n`): rappresenta una continuazione nella riga successiva.

Apici doppi

Racchiudendo una serie di caratteri tra una coppia di apici doppi si mantiene il valore letterale di questi caratteri, a eccezione di: $, ‘, \.

I simboli $ e ‘ (dollaro e apice inverso) mantengono il loro significato speciale all'interno di una stringa racchiusa tra apici doppi, mentre la barra obliqua inversa (\) si comporta come carattere di escape solo quando è seguita da $, ‘, ", e \; quando si trova al termine della riga serve come indicatore di continuazione nella riga successiva.

Si tratta di una particolarità molto importante, attraverso la quale è possibile definire delle stringhe in cui si possono inserire:

- variabili e parametri;

Esempi:

```
$ echo "Il parametro \$0 contiene: \"$0\""(invio)
Il parametro $0 contiene: "-bash"
```

Apici doppi

Racchiudendo una serie di caratteri tra una coppia di apici doppi si mantiene il valore letterale di questi caratteri, a eccezione di: $, ‘, \.

I simboli $ e ‘ (dollaro e apice inverso) mantengono il loro significato speciale all'interno di una stringa racchiusa tra apici doppi, mentre la barra obliqua inversa (\) si comporta come carattere di escape solo quando è seguita da $, ‘, ", e \; quando si trova al termine della riga serve come indicatore di continuazione nella riga successiva.

Si tratta di una particolarità molto importante, attraverso la quale è possibile definire delle stringhe in cui si possono inserire:

- variabili e parametri;
- comandi da sostituire.

Esempi:

```
$ echo "Il parametro \$0 contiene: \"$0\""(invio)
Il parametro $0 contiene: "-bash"
```

Parametri e variabili

Nella documentazione originale di Bash si utilizza il termine «parametro» per identificare diversi tipi di entità:

- parametri posizionali;

Parametri e variabili

Nella documentazione originale di Bash si utilizza il termine «parametro» per identificare diversi tipi di entità:

- parametri posizionali;
- parametri speciali;

Parametri e variabili

Nella documentazione originale di Bash si utilizza il termine «parametro» per identificare diversi tipi di entità:

- parametri posizionali;
- parametri speciali;
- variabili di shell.

Parametri e variabili

Nella documentazione originale di Bash si utilizza il termine «parametro» per identificare diversi tipi di entità:

- parametri posizionali;
- parametri speciali;
- variabili di shell.

Parametri e variabili

Nella documentazione originale di Bash si utilizza il termine «parametro» per identificare diversi tipi di entità:

- parametri posizionali;
- parametri speciali;
- variabili di shell.

In questo documento, per evitare confusioni, si riserva il termine parametro solo ai primi due tipi di entità.

L'elemento comune tra i parametri e le variabili è il modo con cui questi devono essere identificati quando si vuole leggere il loro contenuto: occorre il simbolo $ davanti al nome (o al simbolo) dell'entità in questione, mentre per assegnare un valore all'entità (sempre che ciò sia possibile), questo prefisso non deve essere indicato.

Parametri

Il termine parametro viene utilizzato qui per definire una variabile speciale che può essere solo letta, e rappresenta alcuni elementi particolari dell'attività della shell.

- Come nel caso delle variabili, per fare riferimento al contenuto di un parametro occorre utilizzare il prefisso $ che di per sé non è parte del nome del parametro.

Parametri

Il termine parametro viene utilizzato qui per definire una variabile speciale che può essere solo letta, e rappresenta alcuni elementi particolari dell'attività della shell.

- Come nel caso delle variabili, per fare riferimento al contenuto di un parametro occorre utilizzare il prefisso $ che di per sé non è parte del nome del parametro.
- Tuttavia, per maggiore chiarezza espressiva, dal momento che non è possibile assegnare un valore a queste entità, quando nelle documentazioni si fa riferimento a un parametro, si utilizza quasi sempre il suo nome (o il simbolo che rappresenta) preceduto dal simbolo $.

Parametri

Il termine parametro viene utilizzato qui per definire una variabile speciale che può essere solo letta, e rappresenta alcuni elementi particolari dell'attività della shell.

- Come nel caso delle variabili, per fare riferimento al contenuto di un parametro occorre utilizzare il prefisso $ che di per sé non è parte del nome del parametro.
- Tuttavia, per maggiore chiarezza espressiva, dal momento che non è possibile assegnare un valore a queste entità, quando nelle documentazioni si fa riferimento a un parametro, si utilizza quasi sempre il suo nome (o il simbolo che rappresenta) preceduto dal simbolo $.
- Quindi, dire che il primo parametro posizionale si chiama $1 non è esatto: è semplicemente 1, solo che per leggerne il contenuto si deve aggiungere $ davanti e non esiste la possibilità di trattare questo 1 come una variabile di shell.

Parametri

Il termine parametro viene utilizzato qui per definire una variabile speciale che può essere solo letta, e rappresenta alcuni elementi particolari dell'attività della shell.

- Come nel caso delle variabili, per fare riferimento al contenuto di un parametro occorre utilizzare il prefisso $ che di per sé non è parte del nome del parametro.
- Tuttavia, per maggiore chiarezza espressiva, dal momento che non è possibile assegnare un valore a queste entità, quando nelle documentazioni si fa riferimento a un parametro, si utilizza quasi sempre il suo nome (o il simbolo che rappresenta) preceduto dal simbolo $.
- Quindi, dire che il primo parametro posizionale si chiama $1 non è esatto: è semplicemente 1, solo che per leggerne il contenuto si deve aggiungere $ davanti e non esiste la possibilità di trattare questo 1 come una variabile di shell.
- Inoltre, parlare del parametro 1, può essere fonte di confusione.

Parametri

Il termine parametro viene utilizzato qui per definire una variabile speciale che può essere solo letta, e rappresenta alcuni elementi particolari dell'attività della shell.

- Come nel caso delle variabili, per fare riferimento al contenuto di un parametro occorre utilizzare il prefisso $ che di per sé non è parte del nome del parametro.
- Tuttavia, per maggiore chiarezza espressiva, dal momento che non è possibile assegnare un valore a queste entità, quando nelle documentazioni si fa riferimento a un parametro, si utilizza quasi sempre il suo nome (o il simbolo che rappresenta) preceduto dal simbolo $.
- Quindi, dire che il primo parametro posizionale si chiama $1 non è esatto: è semplicemente 1, solo che per leggerne il contenuto si deve aggiungere $ davanti e non esiste la possibilità di trattare questo 1 come una variabile di shell.
- Inoltre, parlare del parametro 1, può essere fonte di confusione.
- Un parametro è definito, cioè esiste, quando contiene un valore, compresa la stringa vuota.

Parametri posizionali

Un parametro posizionale è definito da una o più cifre numeriche a eccezione del parametro $0 che ha invece un significato speciale. (unità standard di input)

I parametri posizionali rappresentano gli argomenti forniti al comando: `$1` è il primo, $2 è il secondo, e così di seguito.

Quando viene eseguita una funzione, questi parametri vengono rimpiazzati temporaneamente con gli argomenti forniti (vaòori attuali) alla funzione.

Quando si utilizza un parametro composto da più di una cifra numerica, occorre racchiudere questo numero tra parentesi graffe: `${10}`, `${11}`,...

Variabili di shell 1/2

Una variabile è definita quando contiene un valore, compresa la stringa vuota.

L'assegnamento di un valore si ottiene con una dichiarazione del tipo seguente:

```
<nome-di-variabile>=[<valore>]
```

Il nome di una variabile può contenere lettere, cifre numeriche e il segno di sottolineatura, ma il primo carattere non può essere un numero.

Se non viene fornito il valore da assegnare, si intende la stringa vuota.

La lettura del contenuto di una variabile si ottiene facendone precedere il nome dal simbolo $.

Variabili di shell 2/2

In particolare vanno prese in considerazione le variabili descritte di seguito.

`PATH`

È un elenco di directory separato da due punti verticali (`:`).

Il valore predefinito dipende dalla configurazione della shell, e un valore comune potrebbe essere il seguente:

```
/usr/local/bin:/bin:/usr/bin:.:
```

Esempio:

```
$ $PATH
-bash: /anaconda3/bin:/opt/local/bin:/opt/local/sbin:/Users/paoluzzi/\
anaconda3/bin:/Users/paoluzzi/anaconda/bin:/usr/local/bin:/usr/bin:/bin:\
/usr/sbin:/sbin:/Library/TeX/texbin:/opt/X11/bin:/Library/Apple/usr/\
bin: No such file or directory
```

Esportazione

Quando si creano o si assegnano delle variabili, queste hanno una validità limitata all'ambito della shell stessa, per cui, i comandi interni sono al corrente di queste variazioni mentre i programmi che vengono avviati non ne risentono.

Perché anche i programmi ricevano le variazioni fatte sulle variabili, queste devono essere esportate.

L'esportazione delle variabili si ottiene con il comando interno `export`.

Esempi

```
$ PIPPO="ciao"
$ export PIPPO
```

Crea la variabile `PIPPO` e quindi la esporta.

```
$ export PIPPO="ciao"
```

Esegue la stessa operazione dell'esempio precedente, ma in un colpo solo.

Section 3

Bash: sintassi dei comandi

Tipi di comandi

Con il termine «comando» si intendono diversi tipi di entità che hanno in comune il modo con cui vengono utilizzati: attraverso un nome seguito eventualmente da alcuni argomenti.

Può trattarsi dei casi seguenti.

Comandi interni Detti anche comandi di shell, sono delle funzioni predefinite all'interno della shell.

Tipi di comandi

Con il termine «comando» si intendono diversi tipi di entità che hanno in comune il modo con cui vengono utilizzati: attraverso un nome seguito eventualmente da alcuni argomenti.

Può trattarsi dei casi seguenti.

Comandi interni Detti anche comandi di shell, sono delle funzioni predefinite all'interno della shell.

Funzioni Dette anche funzioni di shell, sono funzioni scritte all'interno di uno script di shell.

Tipi di comandi

Con il termine «comando» si intendono diversi tipi di entità che hanno in comune il modo con cui vengono utilizzati: attraverso un nome seguito eventualmente da alcuni argomenti.

Può trattarsi dei casi seguenti.

Comandi interni Detti anche comandi di shell, sono delle funzioni predefinite all'interno della shell.

Funzioni Dette anche funzioni di shell, sono funzioni scritte all'interno di uno script di shell.

Alias Sono dei nomi associati ad altri comandi, di solito con l'aggiunta di qualche argomento.

Tipi di comandi

Con il termine «comando» si intendono diversi tipi di entità che hanno in comune il modo con cui vengono utilizzati: attraverso un nome seguito eventualmente da alcuni argomenti.

Può trattarsi dei casi seguenti.

Comandi interni Detti anche comandi di shell, sono delle funzioni predefinite all'interno della shell.

Funzioni Dette anche funzioni di shell, sono funzioni scritte all'interno di uno script di shell.

Alias Sono dei nomi associati ad altri comandi, di solito con l'aggiunta di qualche argomento.

Tipi di comandi

Con il termine «comando» si intendono diversi tipi di entità che hanno in comune il modo con cui vengono utilizzati: attraverso un nome seguito eventualmente da alcuni argomenti.

Può trattarsi dei casi seguenti.

Comandi interni Detti anche comandi di shell, sono delle funzioni predefinite all'interno della shell.

Funzioni Dette anche funzioni di shell, sono funzioni scritte all'interno di uno script di shell.

Alias Sono dei nomi associati ad altri comandi, di solito con l'aggiunta di qualche argomento.

In maniera semplificata, possono essere visti come un modo diverso per identificare comandi già esistenti.

Programmi Detti anche comandi esterni perché non sono contenuti nella shell che li avvia.

Lista di comandi

La lista di comandi è una sequenza di una o più pipeline di comandi separate da
`;`, `&`, `&&`, `||`, e terminata da `;`, `&`, oppure dal codice di interruzione di riga (`\n`).

Parti della lista sono raggruppabili attraverso parentesi (tonde o graffe) per controllarne la sequenza di esecuzione.

Il valore di uscita della lista corrisponde a quello dell'ultimo comando della stessa lista che ha potuto essere eseguito.

Nelle sezioni seguenti vengono descritti questi operatori.

Separatore di comandi «;»

I comandi separati da un punto e virgola (;) sono eseguiti sequenzialmente.

Il simbolo punto e virgola può essere utilizzato per separare una serie di comandi posti sulla stessa riga, o per terminare una lista di comandi quando c'è la necessità di farlo (per distinguerlo dall'inizio di qualcos'altro).

Idealmente, il punto e virgola sostituisce il codice di interruzione di riga.

Esempi

```
# ./config ; make ; make install
```

Avvia in sequenza una serie di comandi per la compilazione e installazione di un programma ipotetico.

```
$ echo "uno" ; echo "due"
$ echo "uno" ; echo "due" ;
```

I due comandi sono equivalenti: nel secondo la lista viene conclusa con un punto e virgola, ma ciò non produce alcuna differenza di comportamento.

Di seguito si vedono due pezzi di script equivalenti: nel secondo si sostituisce il punto e virgola con un codice di interruzione di riga, dato che il contesto lo consente.

```
$ ls ; echo "Ciao a tutti"
```

Ridirezione

Prima che un comando sia eseguito, si può ridirigere il suo input e il suo output utilizzando una speciale notazione interpretata dalla shell.

La ridirezione viene eseguita, nell'ordine in cui appare, a partire da sinistra verso destra.

Se si utilizza il simbolo < da solo, la ridirezione si riferisce allo standard input (corrispondente al descrittore di file zero.

Se si utilizza il simbolo > da solo, la ridirezione si riferisce allo standard output (corrispondente al descrittore di file numero uno).

La parola che segue l'operatore di ridirezione è sottoposta a tutta la serie di espansioni e sostituzioni possibili.

Se questa parola si espande in più parole viene segnalato un errore.

Descrittore di file

Si distinguono tre tipi di descrittori di file per l'input e l'output:

```
0 = standard input;

1 = standard output;

2 = standard error.
```

Ridirezione dell'input

`[n]< <file>`

La ridirezione dell'input fa sì che il file il cui nome risulta dall'espansione della parola alla destra del simbolo < venga letto e inviato al descrittore di file n, oppure, se non indicato, allo standard input pari al descrittore di file zero.

Esempi

```
$ sort < ./elenco
```

Emette il contenuto del file `elenco` (che si trova nella directory corrente) riordinando le righe. `sort` riceve il file da ordinare dallo standard input.

```
$ sort 0< ./elenco
```

Esegue la stessa cosa dell'esempio precedente, con la differenza che viene indicato esplicitamente il descrittore dello standard input.

Ridirezione normale dell'output 1/2

`[n]> <file>`

La ridirezione dell'output fa sì che il file il cui nome risulta dall'espansione della parola alla destra del simbolo > venga aperto in scrittura per ricevere quanto proveniente dal descrittore di file n, oppure, se non indicato, dallo standard output pari al descrittore di file numero uno.

Di solito, se il file da aprire in scrittura esiste già, viene sovrascritto, sempre che non sia attiva la modalità `noclobber;` (si veda il comando interno `set` (48.31).

Se invece è attiva la modalità `noclobber`, si ottiene l'aggiunta di dati al file eventualmente esistente.

Per garantire la sovrascrittura di un file che potrebbe esistere già, si può utilizzare l'operatore di ridirezione >|.

Ridirezione normale dell'output 2/2

Esempi

```
$ ls > ./dir.txt
```

Crea il file `dir.txt` nella directory corrente e gli inserisce l'elenco dei file della directory corrente.

```
$ ls 1> ./dir.txt
```

Esegue la stessa operazione dell'esempio precedente con la differenza che il descrittore che identifica lo standard output viene indicato esplicitamente.

```
$ ls 1>| ./dir.txt
```

Esegue la stessa operazione del primo esempio, ma si indica in maniera inequivocabile che il file `dir.txt` deve essere creato, anche se è attiva la modalità `noclobber`.

```
$ ls XtgEWSjhy * 2> ./errori.txt
```

Crea il file `errori.txt` nella directory corrente e gli inserisce i messaggi di errore generati da `ls` quando si accorge che il file `XtgEWSjhy` non esiste.

Section 4

Bash: programmazione

Caratteristiche di uno script

Nei sistemi Unix esiste una convenzione attraverso la quale si automatizza l'esecuzione dei file script.

Prima di tutto, uno script è un normalissimo file di testo contenente una serie di istruzioni che possono essere eseguite attraverso un interprete.

Per eseguire uno script occorre quindi avviare il programma interprete e informarlo di quale script questo deve eseguire.

Per esempio, il comando

```
$ bash pippo
```

avvia l'eseguibile `bash` come interprete dello script `pippo`. Per evitare questa trafila, si può dichiarare all'inizio del file script il programma che deve occuparsi di interpretarlo.

Per questo si usa la sintassi seguente:

#!

Quindi, si attribuisce a questo file il permesso in esecuzione.

Quando si tenta di avviare questo file come se si trattasse di un programma, il sistema avvia in realtà l'interprete.

Perché tutto possa funzionare, è necessario che il programma indicato nella prima riga dello script sia raggiungibile così come è stato indicato, cioè sia provvisto del percorso necessario.

Per esempio, nel caso di uno script per la shell Bash (`/bin/bash`), la prima riga sarà la seguente:

```
#!/bin/bash
```

Il motivo per il quale si utilizza il simbolo # iniziale, è quello di permettere ancora l'utilizzo dello script nel modo normale, come argomento del programma interprete: rappresentando un commento non interferisce con il resto delle istruzioni.

Strutture

Per la formulazione di comandi complessi si possono usare le tipiche strutture di controllo e di iterazione dei linguaggi di programmazione più comuni.

Queste strutture sono particolarmente indicate per la preparazione di script di shell, ma possono essere usate anche nella riga di comando di una shell interattiva.

30 Bash Script Examples

Licenza GNU GPL

GNU GENERAL PUBLIC LICENSE - Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Testo originale completo: http://www.fsf.org/copyleft/gpl.html