

Uso di GCC da riga di comando

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

Programmazione

anno accademico 2016–2017

Shell

La **shell** è una interfaccia utente per accedere ai servizi del sistema operativo.

- ▶ In genere, è a riga di comando (Command Line Interface, CLI), ma può anche essere grafica.
- ▶ Tipicamente, consente l'esecuzione di **script**, programmi scritti in un linguaggio legato alla shell che permettono di combinare i comandi di shell e la automatizzazione di certe procedure.

```
C:\temp> dir
Volume in drive C is C
Volume Serial Number is 74F5-B93C

Directory of C:\temp

2009-08-25  11:59    <DIR>          .
2009-08-25  11:59    <DIR>          ..
2007-03-01  11:37           2,321,600 AdobeUpdater12345.exe
2009-04-03  10:01           27,988 dd_depcheckdotnetfx30.txt
2009-04-03  10:01             764 dd_dotnetfx3error.txt
2009-04-03  10:01          32,572 dd_dotnetfx3install.txt
2009-06-09  13:46          35,145 GenProfile.log
2009-08-05  12:11           155 KB968856.log
2009-04-20  08:37           402 MS329e0b.LOG
2009-04-09  16:34          38,895 office111.log
2009-04-03  16:02    <DIR>          OfficePatches
2009-07-14  14:30    <DIR>          OHotfix
2009-08-25  10:52           16,384 PerfLib_Perfdata_c30.dat
2009-04-03  10:01           1,744 uxeventlog.txt
2009-08-25  11:42          50,245,632 WFV2F.tmp
2009-04-20  10:07           1,397 {AC76BA86-7AD7-1033-7B44-A81200000003}.ini
2009-04-20  10:13           617 {AC76BA86-7AD7-1033-7B44-A81300000003}.ini
13 File(s)          52,723,295 bytes
 4 Dir(s)          83,570,208,768 bytes free
```

Comandi utili

La maggior parte dei sistemi hanno i seguenti comandi:

- ▶ contenuto di una directory (folder, cartella): `dir`, `ls`
- ▶ cambiare directory: `cd`
- ▶ tornare alla cartella precedente: `cd ..`
- ▶ creare una directory: `md`, `mkdir`
- ▶ ridirezione dell'output: `>`
 - ▶ pone il risultato di un comando in un file
 - ▶ es: `dir > lista_file.txt`
- ▶ accodamento dell'output: `>>`
 - ▶ aggiunge il risultato in coda al file (non lo riscrive)
- ▶ leggere file di testo: `more`, `less`
- ▶ concatenazione di comandi: `|`

Operazioni preliminari

Trovare `gcc`

- ▶ aprire una shell
- ▶ eseguire `gcc`
- ▶ se il risultato è qualcosa del tipo
`gcc: fatal error: no input files, tutto OK`
- ▶ altrimenti:
 - ▶ assicurarsi di averlo installato
 - ▶ trovarlo (`which`, `get-command`)
 - ▶ aggiungerlo al `path`
 - ▶ `path` è la variabile del sistema operativo che contiene le directory dove vengono cercati i programmi da eseguire
 - ▶ l'istruzione per aggiungere la directory di `gcc` cambia da sistema a sistema
 - ▶ in PowerShell può essere qualcosa tipo:
`path = %PATH%;C:\Programmi\Dev-Cpp\bin`

Precompilazione

Per capire cosa fa la precompilazione, eseguire:

- ▶ `gcc hello1.c -E`
 - ▶ per averlo su file: `gcc hello1.c -E -o hello1.txt`
 - ▶ oppure: `gcc hello1.c -E > hello1.txt`
- ▶ confrontare `hello1.c` con `hello1.txt`

Ricorsività di `#include`

Il precompilatore opera ricorsivamente sui file indicati da `#include`

- ▶ osservare il contenuto di `hello2.c` e `hello2.h`
- ▶ aggiungendo una macro che usa una macro già definita, il risultato non cambia
- ▶ definendo nuovamente una macro già definita, il compilatore dà errore

Compilazione condizionale (`#ifdef` e `#ifndef`)

- ▶ `#ifdef MACRO codice ... #endif` passa il codice al compilatore solo se la macro è stata definita
 - ▶ osservare la presenza di una direttiva condizionale `#ifdef` in `hello3.c`
 - ▶ verificare l'effetto di `#ifdef` cambiando il nome della macro
- ▶ `#ifndef MACRO codice ... #endif` passa il codice al compilatore solo se la macro non è stata definita
 - ▶ spesso usata in testa ai file `.h` per evitare di includerli più volte

Modularizzazione

- ▶ l'uso di `iocrema.c` e `iocrema.h` rende il codice `hello4.c` più leggibile
- ▶ la libreria `iocrema` può essere facilmente riutilizzata in altri programmi

Compilazione

Per precompilare e compilare (prima e seconda fase):

- ▶ `gcc -c hello1.c -o hello1.o`
 - ▶ Se non si specifica l'output, il compilatore produce automaticamente `hello1.o`
- ▶ `gcc -c hello4.c iocrema.c`
 - ▶ `-o` si può utilizzare solo se si compila (`-c`) un singolo file
 - ▶ vengono generati `hello4.o` `iocrema.o`

Collegamento (linking)

Per trasformare gli oggetti in eseguibili:

- ▶ `gcc hello1.o -o hello1.exe`
- ▶ `gcc hello4.o iocrema.o -o hello4.exe`
 - ▶ e se si fornisce un solo oggetto?

Generazione dell'eseguibile in una sola passata

Precompilazione, compilazione e linking:

- ▶ `gcc hello1.c -o hello1.exe`
- ▶ `gcc hello4.c iocrema.c -o hello4.exe`
- ▶ se non si specifica l'output, genera `a.exe` (o `a.out`)

Esercizio

Scrivere un programma C che stampi a video il proprio nome.

Traccia:

1. aprire notepad (`notepad.exe`)
2. aggiungere la funzione `main`
 - ▶ ogni programma C deve averne una (e solo una)
 - ▶ la sua struttura è fissata (la si può copiare da uno dei programmi di esempio)
 - ▶ aggiungere `"return 0;"` alla fine di `main` è sempre una buona abitudine
3. includere `stdio.h`
 - ▶ serve per ogni programma che debba fare I/O
4. includere `iocrema.h`
 - ▶ la procedura `StampaStringa` fa esattamente quello che serve
5. aggiungere una opportuna chiamata di `StampaStringa`
 - ▶ farla seguire da `ACapo` è un tocco di classe
6. salvare il file, compilare ed eseguire