Introduzione a Linux

Lezione 7

Programmare in Linux

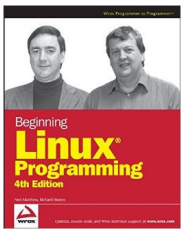
Ruggero Donida Labati

Laboratorio di Sistemi Operativi
Università degli Studi di Milano
Dipartimento di Informatica
A.A. 2024/2025

Materiale prodotto originariamente da Ruggero Donida Labati e Angelo Genovese

Panoramica della lezione

- Verrà introdotta la programmazione di script bash per automatizzare i comandi Linux
- Saranno introdotti gli ambienti di sviluppo principali
- Non verranno spiegati i linguaggi di programmazione



Sommario (1/2)

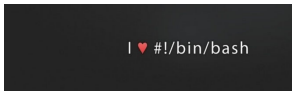
1. Programmazione di script bash
   - Variabili
   - Condizioni, Cicli, Funzioni
   - Input da tastiera
2. Programmare in C/C++
   - Linea di comando
   - Make e makefile
   - Ambienti di sviluppo integrato (IDE)

Sommario (2/2)

3. Programmare in Java
   - Installazione framework
   - Ambienti di sviluppo integrato (IDE)
4. Programmare in PHP
5. Software di elaborazione numerica
   - Matlab
   - Octave
6. Esercizi

1. Programmazione di script bash

1. Introduzione
2. Variabili
3. Condizioni
4. Cicli
5. Funzioni
6. Input da tastiera
7. Operazioni aritmetiche
8. Esecuzione del programma



Introduzione agli script bash (1/2)

- Uno script bash è un file di testo
- Contiene i comandi da eseguire
  - Interpretati in sequenza senza compilazione
- Permette l'uso di comandi tipici dei linguaggi di programmazione
  - Condizioni
  - Cicli
  - Funzioni

Introduzione agli script bash (2/2)

- Lo script inizia sempre con il percorso dell'interprete
  - Es. *#!/bin/sh*
- Lo script più semplice prevede una sequenza di comandi
  - *<Comando_1>*
    <Comando_2>
    Ecc.

Variabili

- Per assegnare un valore ad una variabile si usa il carattere = (senza spazi)
  - *<nome_variabile>=<valore>*
- Se si vuole fare in modo che la variabile venga esportata anche ai processi figli bisogna precedere l'assegnamento con la parola chiave *export*
- Per accedere al contenuto di una variabile si deve precedere il nome con il carattere $
  - Es: echo $ *<nome_variabile>*

Condizioni (1/3)

- Condizione if
  - *if [<espressione_da_valutare]*
    then
    <comando>
    else
    <comando_2>
    fi

Condizioni (2/3)

- Sono possibili diversi operatori per la costruzione di espressioni
  - *-z <var_1>*
    - La variabile *<var_1>* è di lunghezza 0
  - *<var_1> -eq <var_2>*
    - La variabile *<var_1>* è uguale a *<var_2>*
  - *<var_1> -lt <var_2>*
    - La variabile *<var_1>* è minore di *<var_2>*
  - *-f <nome_file>*
    - Vero se *<nome_file>* esiste ed è un file regolare
  - *-x <nome_file>*
    - Vero se *<nome_file>* esiste ed è eseguibile

Condizioni (3/3)

- Esempio
  - *echo INDOVINA IL NUMERO*
    read NUM
    if [$NUM -eq 42]
    then
    echo Hai indovinato
    elif [$NUM -lt 42]
    then
    echo Troppo basso
    else
    echo Troppo alto
    fi

Cicli

1. While
2. Until
3. Case
4. For

`while` and `until` statements

- while <-- if true then loop

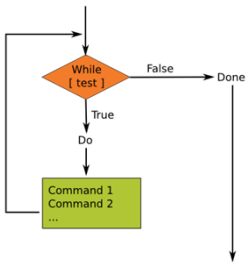
```
while true
do
  echo '\(^o^)/'
done
```

- until <-- if false then loop

```
until false
do
  echo '/(^o^)\'
done
```

Ciclo while (1/2)

- *while [<espressione_da_valutare_vera>]*
  do
  <comando_1>
  <comando_2>
  …
  done



Ciclo while (2/2)

- Esempio
  - *while [$NUM -lt 10]*
    do
    NUM=$(($NUM + 1))
    done

Ciclo until

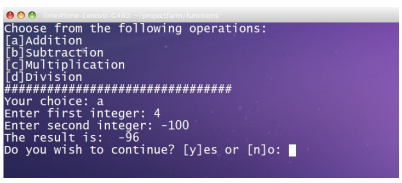
- Simile a *while*

- *until [<espressione_da_valutare_false>]*
  do
  <comando_1>
  <comando_2>
  …
  done

Case (1/2)

- Simile al costrutto *switch*

- *case <var> in*
  <pattern_1>) <comando_1> ;;
  <pattern_2>) <comando_2> ;;
  **) <comando_default>;;*
  esac



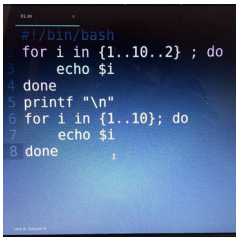
Case (2/2)

- Esempio

```
case $a in
    1) echo sono un 1;;
    2) echo sono un 2;;
    3) echo sono un 3;;
    *) echo Non sono né un 1 né un 2 né un 3;;
esac
```


Ciclo for (1/2)

- *for <var> in <lista_valori>*
  do
    <comando_1>
    <comando_2>
    …
  done



Ciclo for (2/2)

- Esempio
  - *for i in 1 2 3 4 5*
    do
    echo sono $i
    done

Funzioni (1/3)

- È possibile dare un nome a un gruppo di liste di comandi, in modo da poterlo richiamare come si fa per un comando interno normale.
  - *[function] <nome_funzione> () {*
    <comando_1>
    <comando_2>
    }
- Il valore restituito dalla funzione è quello dell'ultimo comando a essere eseguito all'interno

Funzioni (2/3)

- All'interno della funzione possono essere dichiarate delle variabili locali
  - comando *local*
- È possibile utilizzare il comando *return* per concludere anticipatamente l'esecuzione della funzione.

Funzioni (3/3)

- Esempio
  - ```
    #!/bin/bash
    messaggio () {
        echo "ciao,"
        echo "bella giornata vero?"
    }
    messaggio
    ```


    ```

Input da tastiera (1/2)

- Il comando read serve per ottenere l'input da tastiera
  - *read [-p invito] [variabile...]*
- Senza argomenti attende solo la pressione del tasto invio

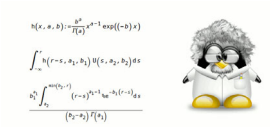
Input da tastiera (2/2)

- Esempio
  - *#!/bin/bash*
    echo -n "Inserisci una frase: "
    read UNO DUE TRE
    echo "La prima parola inserita è --$UNO--"
    echo "La seconda parola inserita è --$DUE--"
    echo "Il resto della frase è --$TRE--"

Operazioni aritmetiche

- È possibile utilizzare operatori aritmetici
  - *+, -, *, /*

- Esempio
  - ```
    #!/bin/bash
    echo "Inserisci un numero"
    read a
    echo " un altro"
    read b
    c = $(($a+$b))
    echo "Il risultato è $c"
    ```



Esecuzione del programma

- Sintassi per l'esecuzione di programmi e script
  - *./<nome_programma_o_script>*


    ```

2. Programmare in C/C++

1. Compilazione a linea di comando
   - Esecuzione
2. Make e makefile
3. Ambienti di sviluppo integrato (IDE)

Compilazione a linea di comando (1/4)

- Il C è un linguaggio compilato
  - I sorgenti sono file di testo .c
- Due step dal sorgente all'eseguibile
  - Compilazione
    - Tramite compilazione si ottengono file oggetto .o
  - Linking
    - Durante il link si uniscono gli oggetti alle librerie .a e si produce l'eseguibile a.out

Compilazione a linea di comando (2/4)

- Il compilatore C standard di Linux è *gcc*
- Il compilatore C++ standard di Linux è *g++*
- La sintassi è simile
  - Compilazione C
    - *gcc -c sorgente.c*
  - Compilazione C++
    - *g++ -c sorgente.c++*
  - Linking C
    - *gcc oggetto1.o oggetto2.o libreria.a -o eseguibile*
  - Linking C++
    - *g++ oggetto1.o oggetto2.o libreria.a -o eseguibile*

Compilazione a linea di comando (3/4)

- Alcune opzioni del compilatore
  - *-L <dir>*
    - Permette di specificare una directory dove cercare le librerie *(/lib, /usr/lib, /usr/local/lib)*
  - *-I <dir> (i maiuscola)*
    - Permette di specificare una directory dove cercare i file .h *(/usr/include, /usr/local/include)*
  - *-l <libreria> (elle minuscola)*
    - Specifica quale libreria dinamica linkare

Compilazione a linea di comando (4/4)

- Esecuzione del programma
  - *./<nome_programma>*

Make e makefile (1/2)

- Strumento che permette di compilare solo i sorgenti modificati tenendo conto delle dipendenze
  - Comando make
- Utilizza per default il file Makefile
  - Contiene i comandi per compilare il progetto e installare l'eseguibile nel sistema

Make e makefile (2/2)

- Vengono specificati gli obiettivi seguiti da ":" e dai file dipendenti
  - Es: *cec++.1 : cec++.1.c*
- Sulla riga successiva, dopo un carattere di tabulazione, si elencano i comandi utili al raggiungimento dell'obiettivo
  - Es:
    CC = gcc
    $(CC) cec++.1.c

Ambienti di sviluppo integrato (IDE) C/C++

- Netbeans
- Eclipse
- CodeBlocks
- KDevelop

3. Programmare in Java

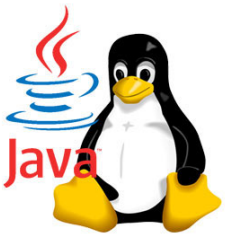
1. Installazione framework
2. Ambienti di sviluppo integrato (IDE)

Installazione del framework

- Come in windows, anche in Linux è necessario installare il framework di sviluppo per Java
- Facilmente eseguibile tramite terminale
  - *sudo add-apt-repository ppa:webupd8team/java*
  - *sudo apt-get update*
  - *sudo apt-get install oracle-java7-installer*
- Controllo dell'installazione
  - *java -version*

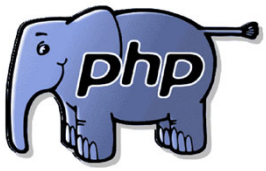
Ambienti di sviluppo integrato (IDE) Java

- Netbeans
- Eclipse
- jEdit



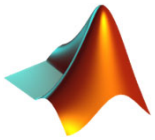
4. Programmare in PHP

- Approccio minimalistico
  - Editor di testo
    - Eventualmente con highlight della sintassi
  - Client FTP
- Ambiente di sviluppo integrato (IDE)
  - Eclipse
  - Netbeans



5. Software di elaborazione numerica

- Matlab
  - Più conosciuto
  - A pagamento



- GNU Octave
  - Quasi completamente compatibile con Matlab
  - Gratis
    - Licenza GPL
  - Installazione tramite terminale
    - *sudo apt-add-repository ppa:octave/stable*
    - *sudo apt-get update*
    - *sudo apt-get install octave*

In sintesi

1. Programmazione di script bash
2. Programmare in C/C++
3. Programmare in Java
4. Programmare in PHP
5. Software di elaborazione numerica

6. Esercizi (1/2)

- Create uno script bash che stampa a video «hello world»
  - È necessario anche impostare i privilegi di esecuzione
- Create uno script che somma due numeri inseriti dall'utente
  - È possibile usare un ciclo per ripetere l'operazione

6. Esercizi (2/2)

- Create un programma C che stampa a video «hello world» ed eseguitelo tramite terminale
- Utilizzate un IDE per compilare il programma
- (*Per chi conosce già Java*) ripetete le operazioni utilizzando il linguaggio Java