# Introduzione a Unix 2

Barbara Guidi

Dipartimento di Informatica
Università di Pisa

Corso Informatica I - 2012/2013

# Outline

# Ricapitolando...

Abbiamo visto alcuni comandi elementari UNIX per navigare il file system (`pwd`, `cd`, `ls`), per creare e cancellare file o directory (`touch`, `mkdir`, `rmdir`, `rm`), e per copiare e spostare file (`cp`, `mv`).

# I metacaratteri (wildcards)

La shell Unix riconosce alcuni caratteri speciali, chiamati **metacaratteri**, che possono comparire nei comandi.

| | |
|---|---|
| `*` | `qualunque sequenza di caratteri` |
| `?` | `qualunque carattere` |

Accade spesso di voler operare su più file contemporaneamente.
Ad esempio, supponiamo di voler copiare tutti i file html di una directory nella sotto-directory html-src.
Usando la wildcard `*` (asterisco) si può scrivere semplicemente:

```
-> cp *.html html-src
```

# Nomi di file e convenzioni

- Esistono precise regole che stabiliscono i nomi con cui possono venire chiamati file e cartelle.
- Nomi con caratteri come / * & %, devono essere evitati per evitare possibili errori di sistema.
- Anche utilizzare nomi composti da parole divise da spazi non è una buona abitudine.
- Nominare file o cartelle usando solo caratteri alfanumerici, lettere e numeri, uniti insieme da _ (underscore) e . (punti).

# Il comando echo

I seguenti esempi usano il comando **echo**, che rimanda sullo schermo ogni parametro.

```
-> echo Ciao!
Ciao!
-> ls
data-new data1 data2 inittab pippo pippo2
-> echo data*
data-new data1 data2
-> echo data?
data1 data2
->
```

# Ridirezione

Di default i comandi Unix prendono l'input da tastiera (**standard input - `stdin`**) e mandano l'output ed eventuali messaggi di errore su video (**standard output - `stdout`, error - `stderr`**). L'input/output in Unix può essere rediretto da/verso file, utilizzando opportuni metacaratteri:

| **Metacarattere** | **Significato** |
|---|---|
| > | ridirezione dell'output |
| >> | ridirezione dell'output (append) |
| < | ridirezione dell'input |
| << | ridirezione dell'input dalla linea di comando |

# Ridirezione - esempi

L'output del comando **echo** viene mandato su **stdout**, che per default è lo schermo. Possiamo ridirigere l'output nel seguente modo:

```
-> echo pippo Topolino > file.txt
-> cat file.txt
pippo Topolino
-> echo e anche Minnie >> file.txt
-> cat file.txt
pippo Topolino e anche Minnie
-> cat list1 list2 > biglist
-> cat biglist
-> sort < biglist
-> sort < biglist > slist
Il contenuto del file biglist viene rediretto verso sort
ed infine come output verso il file slist
```

# Altri comandi utili

Altro comando utile è **find** che ci consente di ricercare file e directory in base al nome, alla data di creazione o alla sua dimensione.
Il comando **find** è molto potente... ma attenzione, talvolta è anche piuttosto lento!!!

Vediamo ora come usare **find**:
Se volessimo cercare in base al nome del file useremo l'opzione **-name**, in questo modo:

```
find /directory/dove/cercare/ -name
termine_da_cercare
```

# Altri comandi utili

- `quota`, mostra lo spazio disco che si ha a disposizione e l'occupazione attuale
- `gzip/gunzip`, compressione/decompressione di file
- `bzip2/bunzip2`, compressione/decompressione di file
- `tar`, creazione di/estrazione da archivio
- `zip/unzip` e `rar/unrar`, servono sempre per la creazione di/estrazione da archivio
- `file <nome>` mostra il tipo del file `<nome>`

# Pipe

Il metacarattere **|** (pipe) serve per comporre **n** comandi "in cascata" in modo che l'output di ciascuno sia fornito in input al successivo. L'output dell'ultimo comando è l'output della pipeline.
`command1 | command2`, l'output dell'esecuzione del primo comando viene passato come input del secondo comando

**Esempio:** `ls | more`
permette di visualizzare l'output di ls pagina per pagina

# I diritti di file e cartelle : chmod

Linux è un sistema multiutente e ogni file/cartella ha un proprietario ed un gruppo di appartenenza insieme ad un elenco di permessi che stabiliscono chi può fare cosa su quel file/cartella.
Usando il comando `'ls -l'` potete vedere i permessi per i file della directory corrente.

```
> ls -l /etc/passwd
-rw-r--r--    1 root     root          981 Sep 20 16:32 /etc/passwd
```

- L'amministratore del sistema (root) ha tutti i permessi (lettura, scrittura, esecuzione) su tutti i file.
- Per le altre categorie di utenti l'accesso ai file è regolato dai permessi
- Ogni file è associato a 9 flag chiamati "**Permission bits**". Il primo carattere (-) serve ad indicare se il file è o non è una directory (in caso lo fosse al posto del trattino c'è d).

# I diritti di file e cartelle : chmod

| User | | | Group | | | Others | | |
|---|---|---|---|---|---|---|---|---|
| R | W | X | R | W | X | R | W | X |

**Read**

- file regolari: possibilità di leggere il contenuto
- directory: possibilità di leggere l'elenco dei file contenuti in una directory

**Write**

- file regolari: possibilità di modificare il contenuto
- directory: possibilità di aggiungere, rimuovere, rinominare file

**Execute**

- file regolari: possibilità di eseguire il file (se ha senso)
- directory: possibilità di fare `cd` nella directory o accedervi tramite path

# I diritti di file e cartelle : chmod

Per cambiare i permessi (potete farlo solo sui file di cui siete proprietari o su quelli su cui avete diritti di scrittura) si usa il comando
**`chmod [ugoa][+-=][rwxXstugo] file`** (man chmod per dettagli).
Esempi:
`chmod ugo+rwx` nomefile assegna diritti completi a tutti (u: user, g: group, o: others) sul file nomefile.
Il comando chmod può essere utilizzato nei seguenti modi:

- chmod con i letterali,
- chmod con i numeri, esempio: `chmod 667 nomefile` (6 indica il permesso di lettura e scrittura e 7 di lettura, scrittura ed esecuzione)

# 2 - Documentazione dei comandi

- `man xxx` : mostra la pagina di 'help' (man = manual) del comando xxx, con istruzioni sull'uso e sulle opzioni possibili (per es. -f, -i, -l ecc...)
- `apropos yyy` : cerca la stringa '`yyy`' nelle pagine di manuale di tutti i comandi Unix. Utile per trovare il nome esatto di un comando che compie l'azione yyy.
- `whatis zzz` : descrive la funzione del comando `zzz`.
- `<comando> --help`

# Outline

# Emacs: informazioni generali

Per lanciare Emacs ed editare il file **primofile**, basta eseguire nella shell il comando

```
-> emacs primofile &
```

Alcune cose da sapere (e sperimentare) su Emacs:

- Effettua un backup del file chiamandolo **primofile~**.
- Quando si modifica su di un file, in realtà si modifica una copia del file tenuta da Emacs in un buffer (nell'esempio, un file chiamato **#primofile#**).
- Le modifiche vanno salvate per renderle permanenti.
- Permette di lavorare su più file alla volta. Non c'è bisogno di mandare in esecuzione più copie di Emacs. Il menu Buffer permette di passare da un file all'altro.
- Un'introduzione all'uso di Emacs si trova nell'Emacs Tutorial, che si può invocare con C-h t oppure dal menu Help

# Compilazione : make e gcc

In Linux, il più diffuso compilatore è GCC (GNU C Compiler):

- GCC è un compilatore portabile, gira attualmente sulle principali piattaforme disponibili al giorno d'oggi ed è in grado di produrre risultati per molti tipi di processore.
- GCC è scritto in C, con una forte attenzione alla portabilità e può compilare se stesso in modo da poter essere adattato a nuovi sistemi con facilità.

A che cosa ci serve un compilatore?
Il compilatore traduce un programma scritto in un linguaggio di alto livello (in questo caso, C) e lo traduce in linguaggio macchina. Un programma "compilato" può dunque essere interpretato ed eseguito dal calcolatore.

# Invocare il gcc

- GCC può essere invocato attraverso il comando "gcc". La modalità più semplice è la seguente: `$ gcc <file sorgente C>`
- `<file sorgente C>` è file contenente un listato di codice sorgente C. Tipicamente, il nome di questi file termina con ".c"
- Il codice che si compila deve contenere una funzione chiamata "`main`", che è l'entry point del programma. In mancanza di questa funzione, il compilatore produce un messaggio di errore.

# Esecuzione

Se il codice sorgente non contiene errori, l'invocazione del comando produce nella directory corrente un file chiamato "`a.out`", che è il risultato della compilazione.
Questo file è un programma eseguibile, che può essere invocato con il comando:

```
$ ./a.out
```

dove "`./`" serve per precisare all'interprete di comandi (shell) che si intende eseguire il programma "`a.out`" nella directory corrente (".").

# Specificare file di output

Quando si invoca GCC è anche possibile specificare un nome per il file di output:

```
$ gcc <file sorgente C> -o <nome file di output>
```

**oppure,**

```
$ gcc -o <nome file di output> <file sorgente C>
```

# Il primo programma: un main vuoto

```
#include <stdio.h>
main() {
}
```

- Abituatevi subito a scrivere le intestazioni correttamente
- Includete sempre almeno `<stdio.h>`
- Salvate il file come `vuoto.c`

# GCC: dettagli

Il comando "`gcc`" accetta molti argomenti ed è estremamente versatile.

`-Wall`, attiva la maggior parte degli warning ("Warning: all")

`-g`, aggiunge informazione per il debug

`-O`, ottimizza il programma compilato

`-pedantic`, forza l'uso di C standard

Per maggiori informazioni: `$ man gcc`

# Compilazione ed esecuzione di vuoto

- `gcc vuoto.c -o vuoto`
- eseguire `./vuoto`, dove "./" serve per precisare all'interprete di comandi (shell) che si intende eseguire il programma nella directory corrente (".").
- Come era lecito attendersi, il risultato è "vuoto"
- Il prompt ritorna immediatamente e senza errori
- Il programma è lecito e l'eseguibile prodotto corretto
- Ovviamente non fa altro che uscire subito dopo essere stato avviato

# Un main sbagliato

```
#include <stdio.h>
main() {
    int my_var
    myvar = my_var + 1;
    printf("my_var e' %d\n", my_var);
}
```

1. Quanti errori ci sono?
2. Quali sono? Salviamo il file come sbagliato.c
3. e proviamo a compilarlo

# Un main sbagliato: leggere gli errori

**Proviamo a compilare il file:**

```
gcc -Wall sbagliato.c -o sbagliato
```

**gcc ci segnala 2 errori:**

- `syntax error before myvar`
- `my_var undeclared`

# Un main sbagliato: correggiamo gli errori

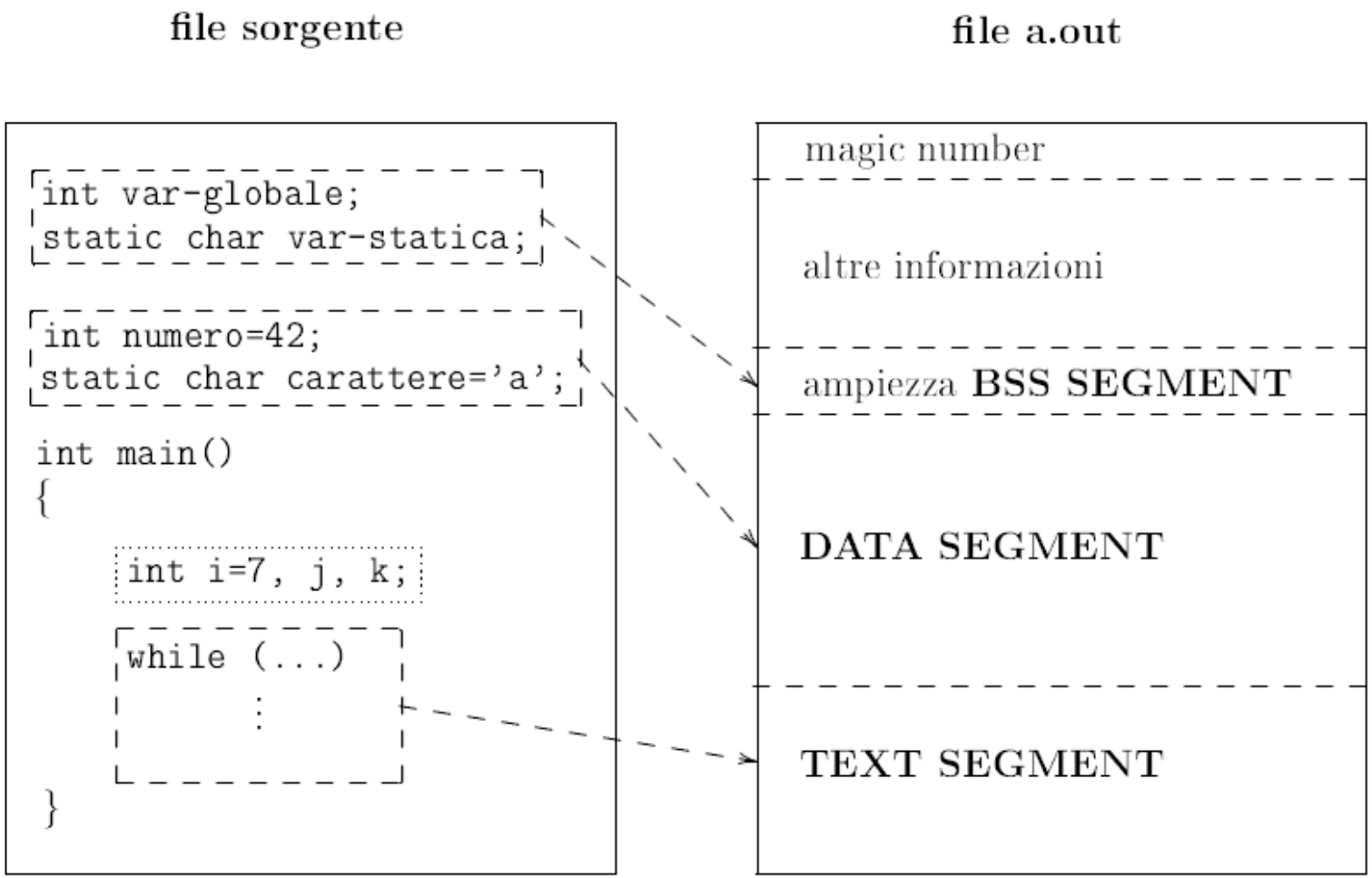
La lista degli errori/warning ci permette di individuare quali sono le righe sbagliate.
Le **warnings** descrivono delle condizioni "inusuali" che possono indicare un problema, sebbene la compilazione possa procedere (e infatti, nel caso vengano riportate solo warnings, il codice compilato viene ugualmente prodotto).
Tutti i programmi dovranno compilare correttamente e senza warning.

```
#include <stdio.h>
main() {
    int my_var = 0;
    my_var = my_var + 1;
    printf("my_var e' %d\n", my_var);
}
```

# Formato del file eseguibile

- La compilazione produce un file eseguibile
- il formato di un eseguibile dipende dal sistema operativo
- In linux un eseguibile ha il formato `ELF` (Executable and Linking Format)
- Il formato `ELF` è leggibile con readelf, objdump, nm
- Un file eseguibile `ELF` è composto da uno o più segmenti. Un segmento è un'area di un file binario in cui si trovano informazioni omogenee, ad esempio il codice del programma, i dati del programma, etc. sono contenuti in dei segmenti separati.

# Formato del file eseguibile (2)



# Formato del file eseguibile (3)

Il comando `size` mostra i segmenti di un eseguibile. Nel caso del programma sviluppato nella sezione precedente, il comando `size esempio` stampa le seguenti informazioni:

```
text    data    bss    dec    hex    filename
1035    232     24     1291   50b    esempio
```

# Approfondimento: printf

La funzione `printf` :

- stampa su standard input (video) dati complessi
- ha un formato articolato, molto potente ma spesso poco chiaro
- ha un numero di opzioni utili e poco conosciute

# printf: esempi d'uso

Scrivere output ben spaziato "| 1| 23|100|":

```
printf("|%3d|%3d|%3d|\n", a, b, c);
```

Scrivere testo e numeri:

```
printf("%lf %c %lf\n", num1, simb, num2);
```

Trasformare un decimale in esadecimale

```
printf("%x", numero);
```

Stampare solo i primi 3 decimali di un float

```
printf("%.3f", numero);
```

# printf

```
int printf (const char *format, ...);
```

Il formato è una stringa costante (non può essere una variabile stringa) che può contenere :

- whitespaces: spazi,tabulazioni, invii a capo (\n)
- caratteri standard (qualunque sequenza di caratteri non bianchi che non inizi per %)
- specificatori di formato (che iniziano per %)

# printf: il formato

Gli specificatori di formato sono così fatti :

```
%[flags][width][.prec][length]type
```

dove :

- `flags`, in qualsiasi ordine, fra: +, spazio, –, # o 0. Specifica cosa premettere ai valori numerici (vedi dettagli su man).
- `width`, specifica il numero minimo di caratteri da stampare per questo argomento.
- `prec`, specifica il numero massimo di caratteri da stampare per questo argomento. Per i floating point, il numero di decimali
- `length`, lunghezza del tipo (vedi dettagli su man).
- `type`, un carattere che specifica il tipo di dato da leggere.

# printf: specificatori di tipo (type)

In uno specificatore di formato il campo obbligatorio type può assumere uno dei seguenti valori:

| **type** | **descrizione** | **argomento** |
|---|---|---|
| `X,x` | Intero esadecimale | `int` |
| `d, i` | Intero decimale: un numero con un segno. | `int` |
| `f, F` | Virgola mobile in notazione normale (fixed point). | `float` |
| `e, E` | Virgola mobile in notazione standard | `float` |
| `g, G` | Virgola mobile in notazione normale o standard | `float` |
| `c` | Singolo carattere | `char` |
| `s` | stringa di caratteri che termina con \0 | `char*` |
| `o` | Intero ottale | `int` |