Sistema Operativo UNIX

Introduzione a UNIX

Il file system

I comandi base

Un sistema di elaborazione dati

Un sistema di elaborazione dati

- dispositivi fisici: (CHIP, alimentatori, memorie)
- software primitivo che controlla i dispositivi del livello inferiore mediante codice di microprogramma (generalmente su ROM). Il set di istruzioni definisce il
- linguaggio macchina: istruzioni elementari per muovere dati, eseguire calcoli e comparare valori; su questo livello i dispositivi I/O sono controllati da valori caricati in speciali registri (registri d'interfaccia)
- sistema operativo vero e proprio, che nasconde la complessità dei livelli inferiori e fornisce al programmatore un insieme di istruzioni di alto livello
- software di sistema: interprete dei comandi (shell), compilatori, editor, interfaccia grafica
- programmi applicativi (Word, Excel, Netscape, ...)

Concetti introduttivi

- Il Software:
  - programmi di sistema (o di base)
  - programmi applicativi

- Il software di base è l'insieme di programmi che rendono facilmente disponibile all'utente le potenzialità offerte dalla macchina (hardware)

  Una parte consistente del software di base è costituito dal

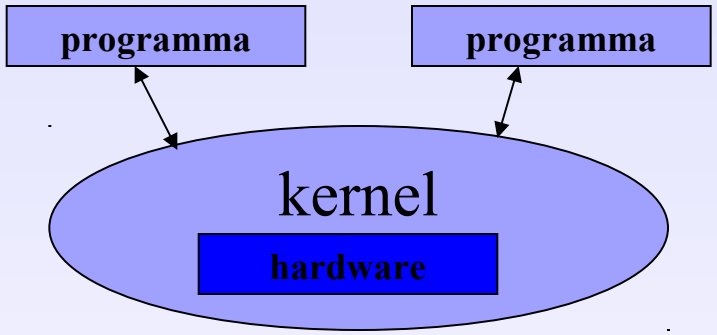
  Sistema Operativo

Sistema operativo

- Il sistema operativo è ciò che regola il funzionamento di tutto l'insieme di queste cose. Volendo schematizzare, si possono distinguere tre aspetti di questo:
  - il kernel;
  - la shell;
  - i programmi di utilità.

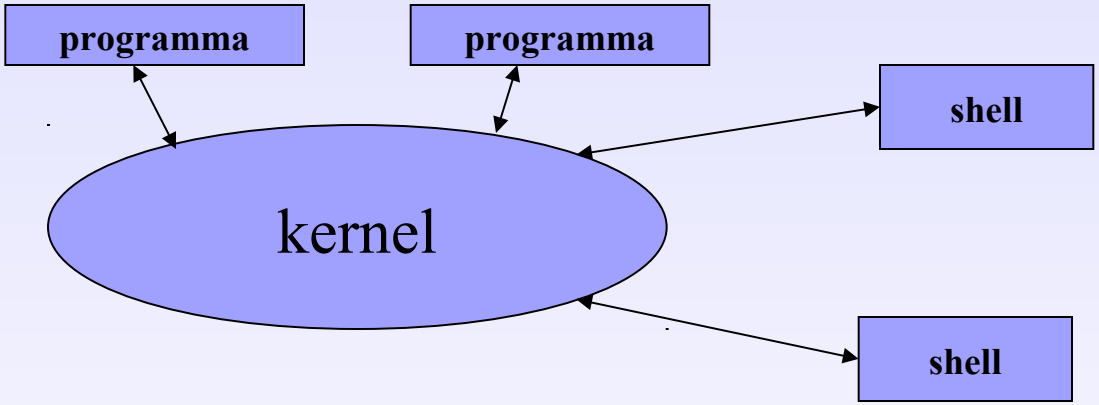
Kernel

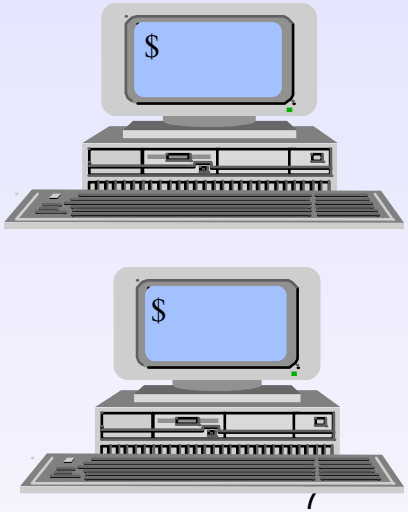
- Il *kernel* è il nocciolo del sistema. Idealmente, è una sorta di astrazione nei confronti delle caratteristiche fisiche della macchina ed è il livello a cui i programmi si rivolgono per qualunque operazione.
  - Ciò significa, per esempio, che i programmi non devono accedere direttamente ai dispositivi fisici, ma possono utilizzare dispositivi logici definiti dal kernel. Questa è la base su cui si fonda la *portabilità* di un sistema operativo su piattaforme fisiche differenti.



Shell

- Il kernel offre i suoi servizi e l'accesso ai dispositivi attraverso chiamate di funzione. Però, mentre i programmi accedono direttamente a questi, perché l'utente possa accedere ai servizi del sistema occorre un programma particolare che si ponga come intermediario tra l'utente (attraverso il terminale) e il kernel.
- Questo tipo di programma è detto *shell*. Come suggerisce il nome (conchiglia), si tratta di qualcosa che avvolge il kernel, come se quest'ultimo fosse una perla.





Shell

- La shell è il programma che consente all'utente di accedere al sistema. I terminali attraverso cui si interagisce con la shell sono comunque parte dell'hardware controllato dal kernel.
- Un programma shell può essere qualunque cosa, purché in grado di permettere all'utente di avviare, e possibilmente di controllare i programmi.
- La forma più semplice, e anche la più vecchia, è la riga di comando presentata da un invito, o *prompt*. Questo sistema ha il vantaggio di poter essere utilizzato in qualunque tipo di terminale.
- Nella sua forma più evoluta, può arrivare a un sistema grafico di icone o di oggetti grafici simili, oppure ancora a un sistema di riconoscimento di comandi in forma vocale. Si tratta sempre di shell.
- Nel seguito si farà riferimento alla bash shell di Unix

Programmi di utilità

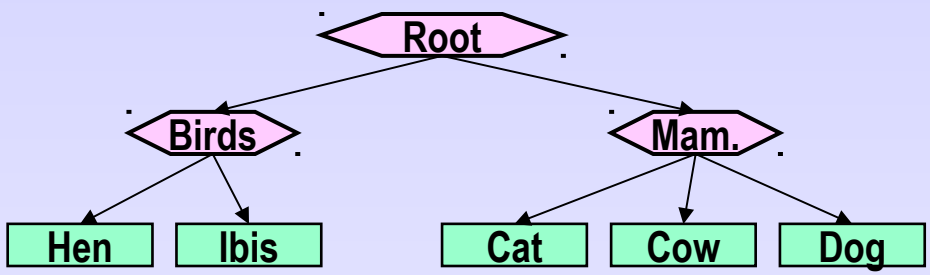
- I programmi di utilità sono un insieme di piccole applicazioni utili per la gestione del sistema. Teoricamente, tutte le funzionalità amministrative per la gestione del sistema potrebbero essere incorporate in una shell; in pratica, di solito questo non si fa. Dal momento che le shell tradizionali incorporano alcuni comandi di uso frequente, spesso si perde la cognizione della differenza che c'è tra le funzionalità fornite dalla shell e i programmi di utilità.
- Programmi applicativi: l'elaboratore non può essere una macchina fine a se stessa. Deve servire a qualcosa, al limite a giocare. È importante ricordare che tutto nasce da un bisogno da soddisfare. I programmi applicativi sono quelli che (finalmente) servono a soddisfare i bisogni, e quindi, rappresentano l'unica motivazione per l'esistenza degli elaboratori.

Il File

- Dal punto di vista dell'utente un file è un insieme di dati correlati che si conservano nel tempo, associato ad un nome
- Dal punto di vista del sistema operativo, un file è un insieme di byte conservato in un dispositivo di memorizzazione
- Il nome:
  - è una sequenza (limitata) di caratteri
  - l'insieme di caratteri leciti dipende dal sistema operativo
  - Unix distingue fra lettere maiuscole e minuscole
- Una directory è un file speciale, gestito direttamente dal sistema operativo, che contiene informazioni relativa a un insieme di file correlati logicamente fra di loro

File system

- Il filesystem è il sistema che organizza i file all'interno dei dispositivi di memorizzazione.



- Generalmente, si utilizzano elenchi strutturati (le directory), per cui da un elenco si viene rimandati a un altro elenco più dettagliato che può contenere l'indicazione di ciò che si cerca o il rinvio a un altro elenco ancora.
- Per questo motivo, la struttura di un filesystem assume quasi sempre una forma ad albero nella quale c'è un'origine a partire da cui si diramano tutti i file. Le diramazioni possono svilupparsi in modo più o meno esteso, a seconda delle esigenze.

File system

- Data l'esistenza di questo tipo di organizzazione, si utilizza una notazione particolare per indicare un file all'interno di un filesystem. Precisamente si rappresenta il percorso necessario a raggiungerlo:
  - una barra obliqua rappresenta la directory principale, altrimenti chiamata anche radice, o root;
  - un nome può rappresentare indifferentemente una directory o un file;
  - un file o una directory che discendono da una directory precedente, si indicano facendo precedere una barra obliqua.
- Per esempio, /uno/due/tre rappresenta il file (o la directory) tre che discende da due, che discende da uno, che a sua volta discende dall'origine
- Esiste quindi una corrispondenza biunivoca fra i nomi dei file e i file in un file system

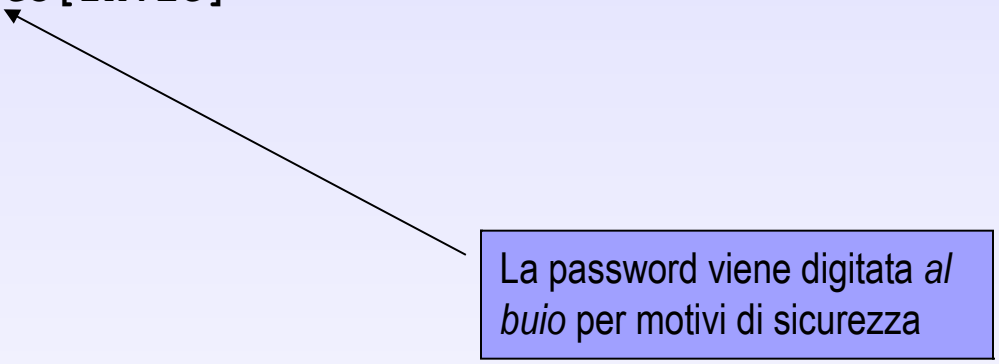
File system

- Utilizzare sempre il nome completo (path name assoluto) di un file può essere scomodo, perciò è stato introdotto il concetto di directory corrente
  - È la directory in cui logicamente un utente si trova in quel particolare momento
  - Tradizionalmente si utilizzano due nomi convenzionali per poter fare riferimento alla directory in cui ci si trova e a quella superiore nella struttura ad albero:
  - . un punto singolo rappresenta la directory in cui ci si trova;
  - .. due punti la directory *padre*.
- I file perciò possono anche essere riferiti tramite un nome relativo (path relativo)
  - Implicitamente il nome viene concatenato con il nome della directory corrente
  - Un path assoluto inizia sempre con il carattere '/'
  - Un path relativo con un qualunque altro carattere

Accesso al sistema

- Per accedere ad un sistema Unix occorre utilizzare un processo di identificazione. Per poter essere correttamente identificati e accedere alle risorse del sistema occorre essere stati preventivamente registrati in un'utenza, rappresentata in pratica da un nominativo-utente e da una password.
  - Si ricorda che con Unix lettere maiuscole o minuscole sono riconosciute come differenti.
- Supponiamo che io possa accedere tramite il nome utente **`pippo`** e password **`pluto`**. L'accesso avverrà tramite un procedimento simile:

```
login: pippo[Invio]
Password: pluto[Invio]
```



Accesso al sistema

- Se si sbaglia qualcosa nella fase di login, viene segnalato un messaggio di errore e si deve ricominciare.
- Dopo avere superato correttamente il login si entra nel sistema. Se si lavora in un ambiente grafico apparirà una finestra con un prompt lampeggiante che sta a indicare che la shell è in attesa di comandi da parte dell'utente.
- Il prompt sarà rappresentato negli esempi seguenti da un simbolo dollaro, ma può essere anche il simbolo di percento o una sequenza di caratteri complessa definita dall'utente.
- Al termine della sessione di lavoro occorrerà scollegarsi.

Gestione della password

- La prima regola per una password sicura consiste nel suo aggiornamento frequente. Quando si cambia la password, viene richiesto inizialmente l'inserimento della password precedente, quindi si può inserire quella nuova, per due volte, in modo da prevenire eventuali errori di battitura.
- L'utente comune può cambiare la propria password, solo la propria. Nell'esempio, l'utente è tizio.

```
$ passwd[Invio]
```

- Prima di accettare una nuova password, viene richiesta quella vecchia.

```
(current) UNIX password: pluto[Invio]
```

- Quindi viene richiesta quella nuova per due volte consecutive.

```
New UNIX password: albero[Invio]
Retype password: albero[invio]
```

Ovviamente le password vecchia e nuova non vengono presentate

- Se la nuova password non viene scritta due volte allo stesso modo, il cambiamento non viene accettato

Directory personale

- Ogni utente, in un sistema UNIX, ha una directory personale, detta anche *home directory*, destinata a contenere tutto ciò che riguarda l'utente.
- Alcune shell interpretano il carattere tilde (~), all'inizio di un percorso, come il nome della directory personale dell'utente.
  - Nello stesso modo, un nome utente preceduto da un carattere tilde, viene sostituito dal nome della directory personale dell'utente stesso.

```
$ echo ~
/home/pippo
$ echo ~pluto
/home/pluto
```

- Quando un utente si collega ad un sistema si posiziona automaticamente nella sua home directory che diventa quindi la sua directory corrente

Comandi

- L'istruzione tipo di Unix è la seguente:
  - (comando) (opzioni) (argomenti)
- dove:
  - (comando) è l'istruzione data
  - (opzioni) sono "variazioni" al comando stesso
  - (argomenti) sono generalmente i file su cui deve intervenire il comando (i file possono ovviamente essere indicati sia con il path assoluto che relativo)
- Un comando importante è man che permette di consultare il manuale del comando specificato
  - In particolare, il comando:

    ```
    $ man man
    ```
  - permette di consultare il manuale del comando man.
- In molti sistemi i comandi accettano l'opzione --help che dà informazioni sull'uso del comando stesso
  - Ad esempio:

    ```
    $  man --help
    ```

Comando echo

- Stampa a video un messaggio
- Uso: echo <messaggio>

```
$  echo Hello world [Invio]
Hello world
```

- Posso eseguire più comandi sulla stessa riga di comando, purché separati da ';'.

```
$ echo Ciao ; echo Ciao [Invio]
Ciao
Ciao
```

- Opzioni principali:

  -n il cursore non va a capo dopo il messaggio

```
$ echo –n Ciao ; echo Ciao [Invio]
CiaoCiao
```

Comando cd

- Cambia la directory corrente
- Uso: cd [nome-directory]
  - In assenza di parametri torno nella directory '*home*'
  - Altrimenti mi sposto nella directory specificata

```
$ cd /tmp [Invio]
```
mi sposto nella directory /tmp

```
$ cd [Invio]
```
torno nella mia directory home

```
$ cd ..[Invio]
```
risalgo nella directory padre nell'albero del file system

- A volte può essere necessario sapere qual è la directory corrente, posso usare il comando pwd (print working directory)

```
$ pwd [Invio]
```
(stampo il nome della directory corrente)

- Per creare una directory si usa il comando mkdir nuova-directory

```
$ mkdir sorgenti
```
La nuova directory è una sottodirectory della directory corrente

```
$ mkdir /tmp/prova
```
La nuova directory è una sottodirectory di /tmp

Comando rm

- Rimuove uno o più file
- Uso: rm [opzioni] [nomi-file]
- Opzioni principali
  - -i chiede conferma prima di ogni operazione

- Per eliminare una directory si usa invece **`rmdir`**
  - L'operazione è possibile solo se la directory è vuota

```
$ rm pippo pippo1 /tmp/pluto [Invio]
```

Ho cancellato 3 file

Attenzione! Una volta eliminato un file non è più recuperabile.

Comando ls

- Mostra informazioni relative a uno o più file
- Uso: ls [opzioni] [nomi-file]
  - In mancanza di argomenti mostra il contenuto della directory corrente (si usa il parametro implicito '.')
  - Se l'argomento è un file vengono mostrate le informazioni relative (normalmente solo il nome)
  - Se l'argomento è una directory vengono mostrate le informazioni relative a tutti i file della stessa
- Opzioni principali
  - **`-l`** mostra una informazione completa sui file
  - **`-a`** mostra anche i file *nascosti* (normalmente i file il cui nome inizia con '.' non vengono mostrati)

```
$ ls
Ordine.class  Ordine.java  Ordini.class  Ordini.java
$ ls .
Ordine.class  Ordine.java  Ordini.class  Ordini.java
```

Comando ls

```
% ls -l
-rw-r--r--    1 luca     staff        1846 Feb  8 10:59 Ordine.class
-rw-r--r--    1 luca     staff        2143 Feb  8 10:59 Ordine.java
-rw-r--r--    1 luca     staff        1959 Feb  8 10:59 Ordini.class
-rw-r-----    1 luca     staff        2443 Feb  8 10:59 Ordini.java
```

```
% ls -la
drwxr-xr-x    4 luca     staff         512 Apr  9 15:00 .
drwxr-xr-x    9 luca     staff         512 Apr  9 15:00 ..
-rw-r--r--    1 luca     staff        1846 Feb  8 10:59 Ordine.class
-rw-r--r--    1 luca     staff        2143 Feb  8 10:59 Ordine.java
-rw-r--r--    1 luca     staff        1959 Feb  8 10:59 Ordini.class
-rw-r-----    1 luca     staff        2443 Feb  8 10:59 Ordini.java
```

Si è ottenuta una informazione più dettagliata con il tipo di file, i permessi, l'appartenenza all'utente, al gruppo, la dimensione del file e la data di ultima modifica.

Permessi sui file

- I primi caratteri risultanti dal comando ls –l indicano i permessi sui file
  - Il primo carattere indica le caratteristiche del file
    d indica che il nome corrisponde a una directory
    - caratterizza un file normale
  - Seguono 3 terne di caratteri che corrispondono al permesso di lettura (r), scrittura (w) e esecuzione (x). Il carattere – indica che il permesso corrispondente non è concesso
  - La prima terna fa riferimento all'utente proprietario del file, la seconda al gruppo, la terza a tutti gli altri utenti

```
-rw-r-----    1 luca     staff        2443 Feb  8 10:59 Ordini.java
```

  - Il file Ordini.java può essere letto e modificato dall'utente luca, può essere letto dagli utenti appartenenti al gruppo staff, tutti gli altri utenti non possono utilizzarlo in nessun modo

Permessi sulle directory

- Sulle directory i permessi di lettura, scrittura e esecuzione hanno un significato particolare.
  - Togliendo il permesso in lettura si impedisce la lettura del contenuto della directory, cioè si impedisce l'esecuzione di un comando come ls, mentre l'accesso ai file continua a essere possibile (purché se ne conoscano i nomi).
  - Togliendo il permesso di esecuzione si impedisce di accedere alla directory.
  - Togliendo il permesso di scrittura si impedisce di creare e rimuovere file (cioè di modificare il contenuto della directory).

Comando chmod

- Modifica i permessi relativi ad un file
- Uso: chmod permessi files
- Esempi:
  - $ **chmod +r pippo [*Invio*]** (concede il permesso di lettura al file)
  - $ **chmod -w pippo [*Invio*]** (toglie il permesso di lettura al file)
  - I permessi si possono indicare anche tramite 3 cifre ottali
  - La prima cifra si riferisce al proprietario, la seconda al gruppo, la terza agli altri utenti
  - Il primo bit della cifra ottale (peso 4) al permesso di lettura, il secondo (peso 2) al permesso di scrittura, il terzo (peso 1) al permesso di esecuzione
    - $ **chmod 754 pippo [*Invio*]**
    - $ **ls –l pippo [*Invio*]**

```
-rwxr-wr--    1 luca     staff     2443 Feb  8 10:59 pippo
```

Comando mv

- Sposta o cambia nome ad un file
- Uso:

  `mv nome-corrente nome-nuovo`

  oppure

  `mv files directory`

  – Nel primo caso cambia nome al file
    - $ **`mv pippo pippo1 [`*`Invio`*`]`**
    - $ **`mv pippo /tmp/pippo2 [`*`Invio`*`]`**
  – Nel secondo sposta tutti i file nella directory indicata come ultimo argomento mantenendone il nome
    - $ **`mv pippo pippo1 /tmp [`*`Invio`*`]`**
    - $ **`mv pluto /tmp [`*`Invio`*`]`**
- Opzioni principali:

  -i chiede conferma prima di sovrascrivere un file esistente (il nuovo nome corrisponde ad un file già esistente)
- Il comando mv non può cambiare una serie di nomi in modo sistematico. Non si può cambiare *.mio in *.tuo.

Comando cp

- Copia uno o più file
- Uso:

  `cp nome-corrente nome-copia`

  oppure

  `cp files directory`

  – Nel primo caso crea una copia del file “nome-corrente” che ha nome “nome-copia”

    $ **`cp pippo pippo1 [`*`Invio`*`]`**

    $ **`cp pippo /tmp/pippo2 [`*`Invio`*`]`**

  – Nel secondo copia tutti i file nella directory indicata come ultimo argomento, mantenendone il nome

    $ **`cp pippo pippo1 /tmp [`*`Invio`*`]`**

    $ **`cp pluto /tmp [`*`Invio`*`]`**

- Opzioni principali:

  `-i` chiede conferma prima di sovrascrivere un file esistente (il nuovo nome corrisponde ad un file già esistente)

Creazione di un file

- Esistono vari modi per creare un file. Il modo più semplice per creare un file vuoto è quello di usare il comando touch

  $ **touch pippo[Invio]**

- Dopo aver usato il comando touch per creare il file pippo non si ottiene alcuna conferma dell'avvenuta esecuzione dell'operazione. Questo atteggiamento è tipico dei sistemi Unix i cui comandi tendono a non manifestare il successo delle operazioni eseguite
- Se il file esiste ne viene modificata la data di ultimo aggiornamento

```
$ ls -al
drwxr-xr-x    2 luca     staff       512 Jan 27 15:58 .
drwxr-xr-x    3 luca     staff       512 Jan 27 15:58 ..
-rw-r--r--    1 luca     staff      6098 Sep 25  2002 .Xmodmap
$ date
Sun Apr 11 11:07:57  2004
$ touch .Xmodmap pippo
$ ls -al
drwxr-xr-x    2 luca     staff       512 Jan 27 15:58 .
drwxr-xr-x    3 luca     staff       512 Jan 27 15:58 ..
-rw-r--r--    1 luca     staff      6098 Apr 11 11:08 .Xmodmap
-rw-r--r--    1 luca     staff         0 Apr 11 11:08 pippo
```

Contenuto dei file

- Anche il contenuto dei file può essere analizzato soprattutto quando si tratta di file di testo. Per visualizzare il contenuto di file di testo si utilizzano generalmente i comandi cat e more.

```
$ cat /etc/fstab[Invio]
/dev/hda3    /           ext2      defaults        1  1
/dev/hda2    none        swap      sw
proc         /proc       ignore
/dev/hda1    dos         vfat      quiet,umask=000
/dev/hdc     /mnt/cdrom  iso9660   ro,user,noauto
```

- Con il comando **cat** si è ottenuta la visualizzazione del contenuto del file /etc/fstab, che ovviamente cambia a seconda della configurazione del proprio sistema
- **cat** non si presta alla visualizzazione di file di grandi dimensioni. In tal caso si preferisce usare **more**.

Caratteri jolly: asterisco

- L'asterisco rappresenta una sequenza indefinita di zero o più caratteri, esclusa la barra di separazione tra le directory. Per cui, l'asterisco utilizzato da solo rappresenta tutti i nomi di file disponibili nella directory corrente.

  $ **ls[*Invio*]**

- Il comando ls appena mostrato serve a elencare tutti i nomi di file e directory contenuti nella directory corrente.

  $ **ls *[*Invio*]**

- Questo comando è un po' diverso, nel senso che la shell provvede a sostituire l'asterisco con tutto l'elenco di nomi di file e directory contenuti nella directory corrente.
- Attenzione all'uso del carattere * con rm

  $ **rm *.java [*Invio*]** (cancello tutti i file con estensione .java)

  – Ma se per sbaglio inserisco uno spazio e scrivo

  $ **rm * .java [*Invio*]** (cancello tutti i file e il file .java)

  – Si consiglia di usare sempre **rm –i**

Interrogativo

- Il punto interrogativo rappresenta esattamente un carattere qualsiasi.
- Supponiamo esistano i file **`xy123j4 xy456j5 xy789j111 xy78j67`**

```
$ echo ?????j?[Invio]
xy123j4 xy456j5
$ echo *j*[Invio]
xy123j4 xy456j5 xy789j111 xy78j67
```

Stampo i nomi di tutti i file di esattamente sette caratteri e che contengono la lettera j nella sesta posizione.

Stampo i nomi di tutti i file che contengono la lettera j

Parentesi quadre

- Le parentesi quadre vengono utilizzate per delimitare un elenco o un intervallo di caratteri. Rappresentano un solo carattere tra quelli contenuti, o tra quelli appartenenti all'intervallo indicato.

```
$ echo xy????[4567]*[Invio]
xy123j4 xy456j5
```

- Il comando appena indicato era stato scritto in modo da fornire a echo, come argomento, l'elenco di tutti i file i cui nomi iniziano per xy, proseguono con quattro caratteri qualunque, quindi contengono un carattere da 4 a 7 e terminano in qualunque modo. Lo stesso risultato si poteva ottenere indicando un intervallo nelle parentesi quadre.

```
$ echo xy????[4-7]*[Invio]
```

```
$ echo [A-Z]*
```

Stampo i nomi di tutti i file che iniziano con lettera maiuscola

Ridirezione

- La ridirezione dirotta i dati in modo di destinarli a un file o di prelevarli da un file.

  $ ls –l > elenco[*Invio*]

- Questo comando genera il file elenco con il risultato dell'esecuzione di ls. Si può controllare il contenuto di questo file con cat.

  $ cat elenco[*Invio*]

- Anche l'input può essere ridiretto, quando il comando al quale si vuole inviare è in grado di riceverlo. cat è in grado di emettere ciò che riceve dallo standard input.

  $ cat < elenco[*Invio*]

- Si ottiene in questo modo la visualizzazione del contenuto del file elenco, esattamente nello stesso modo di prima, quando questo nome veniva indicato semplicemente come argomento di cat. Ma adesso lo si invia attraverso lo standard input per mezzo dell'attività della shell.

Ridirezione

- La ridirezione dell'output, come è stata vista finora, genera un nuovo file ogni volta, eventualmente sovrascrivendo ciò che esiste già con lo stesso nome. Sotto questo aspetto, la ridirezione dell'output è fonte di possibili danni.
- La ridirezione dell'output può essere fatta in aggiunta, creando un file se non esiste, o aggiungendovi i dati se è già esistente.

```
$ ls -l /tmp  >>  elenco[Invio]
```

- In tal modo viene aggiunto al file elenco l'elenco dettagliato del contenuto della directory /tmp/.

```
$ cat elenco[Invio]
```

Dispositivi

- Nei sistemi Unix, i dispositivi sono manifestati da file speciali collocati nella directory /dev/.
- Il dispositivo /dev/null corrisponde in lettura a un file vuoto, e in scrittura a una sorta di pozzo senza fondo: tutto ciò che vi viene scritto è perduto. Questa particolarità è molto utile negli script in cui si vuole evitare che i comandi contenuti emettano segnalazioni all'utente.

```
$ ls /bin > /dev/null[Invio]
```

- Il comando appena mostrato non emette nulla sullo schermo perché tutto viene ridiretto verso /dev/null. Si può verificare che in questo file non ci sia più alcuna traccia con il comando seguente:

```
$ cat /dev/null[Invio]
```

- Non si ottiene alcun output.

Ridirezione

- Nell'interazione con l'utente si distinguono tre flussi di informazioni
  - Standard input (identificato con 0, dall'utente al sistema, normalmente la tastiera)
  - Standard output (identificato con 1, dal sistema all'utente, normalmente il monitor)
  - Standard error (identificato con 2, i messaggi di errore, dal sistema all'utente, normalmente il monitor)

```
$  touch qui qua
$  ls qui quo qua
ls: quo: No such file or directory
qua     qui
$  ls qui quo qua > /dev/null
ls: quo: No such file or directory
$  ls qui quo qua 2> errori
qua     qui
$  cat errori
ls: quo: No such file or directory
```

Ho creato i file qui e qua (non quo)

Appare a video un mesaggio di errore

Appare a video solo il mesaggio di errore

I messaggi di errore vengono salvati nel file errore

Pipeline

- La pipeline è una forma di ridirezione in cui la shell invia l'output di un comando come input del successivo.

```
$ cat elenco | sort[Invio]
```

- In questo modo, cat legge il contenuto del file elenco, ma invece di essere visualizzato sullo schermo, viene inviato dalla shell come standard input di sort che lo riordina e poi lo emette sullo schermo.
- Una pipeline può utilizzare anche la ridirezione, per cui, il comando visto precedentemente può essere trasformato nel modo seguente,

```
$ cat < elenco | sort[Invio]
```

- Quando si hanno molti errori di compilazione può essere utile:

```
$  gcc nome.c 2>&1 | head -10
```

Rimando lo standard error sullo standard output poi passo tutto come input del comando head, che mostra solo le prime 10 righe

Comando ps

- Un processo è un programma in esecuzione.
- Il comando ps permette di visualizzare i processi in esecuzione.

```
$ ps[Invio]
  PID      TTY   UID     STIME COMMAND
 3540      con 1005  14:10:21 /usr/bin/bash
 4080      con 1005  18:32:23 /usr/bin/bash
  484      con 1005  18:32:34 /usr/bin/ls
 3916      con 1005  18:32:39 /usr/bin/ps
```

- Si può forzare la fine di un processo tramite il comando kill, specificando il numero del processo.

```
$ kill 484[Invio]
```

history

- !! Riesegue l'ultimo comando
- !stringa Riesegue l'ultimo comando che inizia con stringa

```
$ !gcc
```

Riesegue l'ultimo comando che inizia con gcc

- !numero Riesegue l'ennesimo comando

```
$ !12
```

Riesegue il dodicesimo comando

- history mostra i comandi eseguiti
- !$ l'ultimo parametro del comando precedente
- !* tutti i parametri del comando precedente

```
$  echo Hello world
Hello world
$  echo !*
Hello world
$  echo Ciao !$
Ciao world
```

Il compilatore

- Il compilatore tradizionale di UNIX è cc (c compiler)
- Nei sistemi Linux è generalmente disponibile il gcc (gnu c compiler)
  - Normalmente è accettato anche cc, ma rimanda a gcc

```
$ ls -l
-rw-rw-r-- 1 user user   94 Oct  4 10:59 hello.c
$ gcc hello.c
$ ls -l
-rwxrwxr-x 1 user user 7159 Oct  4 10:59 a.out
-rw-rw-r-- 1 user user   94 Oct  4 10:59 hello.c
```

Il compilatore

- Generalmente è preferibile specificare il nome del file prodotto ed un elenco completo di messaggi

```
$ gcc -Wall hello.c -o hello
$ ls -l
-rwxrwxr-x 1 user user 7159 Oct  4 10:59 a.out
-rwxrwxr-x 1 user user 7159 Oct  4 11:09 hello
-rw-rw-r-- 1 user user   94 Oct  4 10:59 hello.c
```

- Si può alche provare il comando **`make`**

```
$ make hello
cc hello.c -o hello
```

- Si noti che se si ripete il comando, la compilazione non viene ripetuta

```
$ make hello
make: `hello' is up to date.
```

Il compilatore

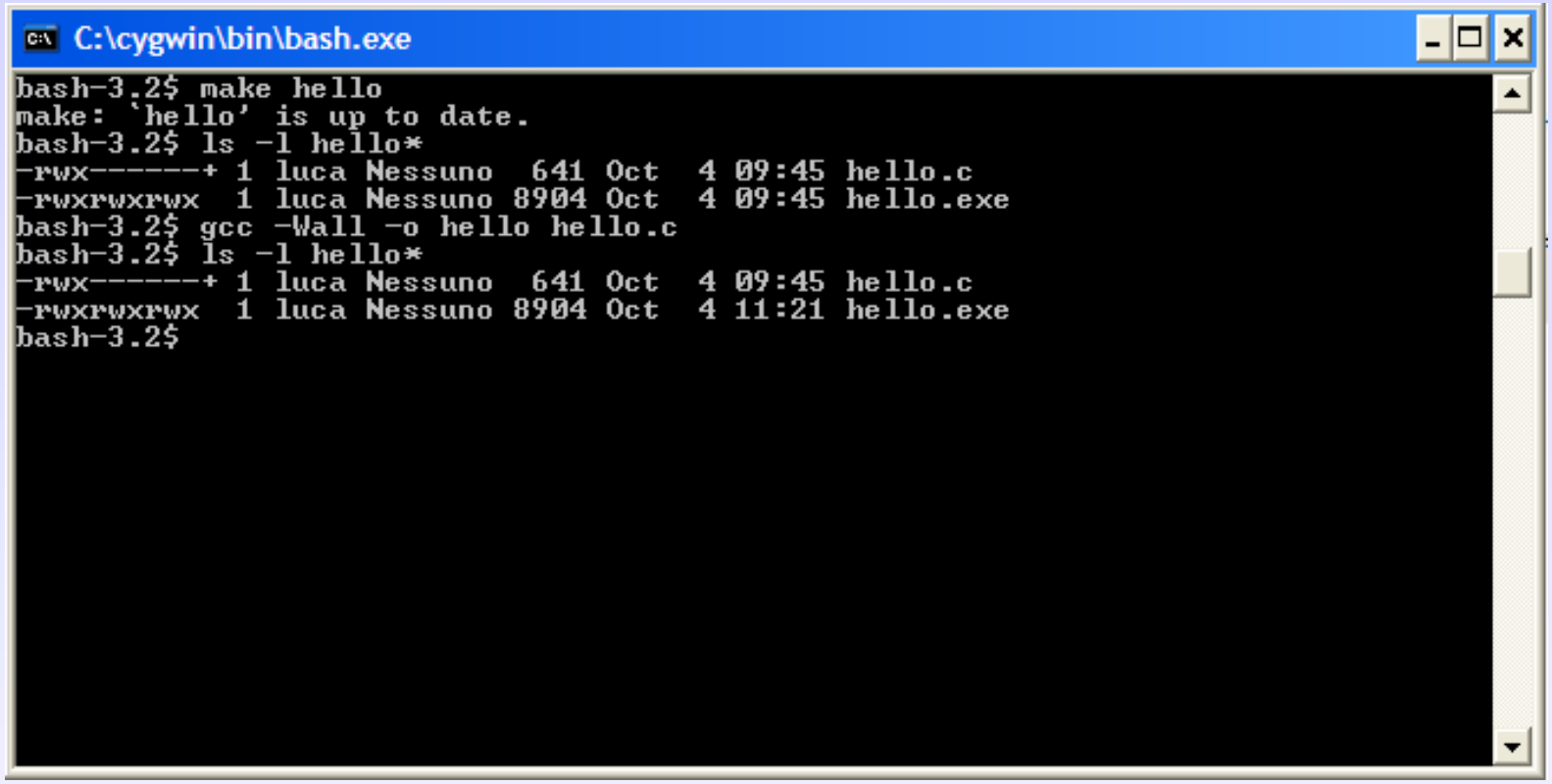
- Attenzione:

```
$ gcc -Wall hello.c -o hello.c
$ ls -l
-rwxrwxr-x 1 user user 7159 Oct  4 10:59 a.out
-rwxrwxr-x 1 user user 7159 Oct  4 11:19 hello.c
```

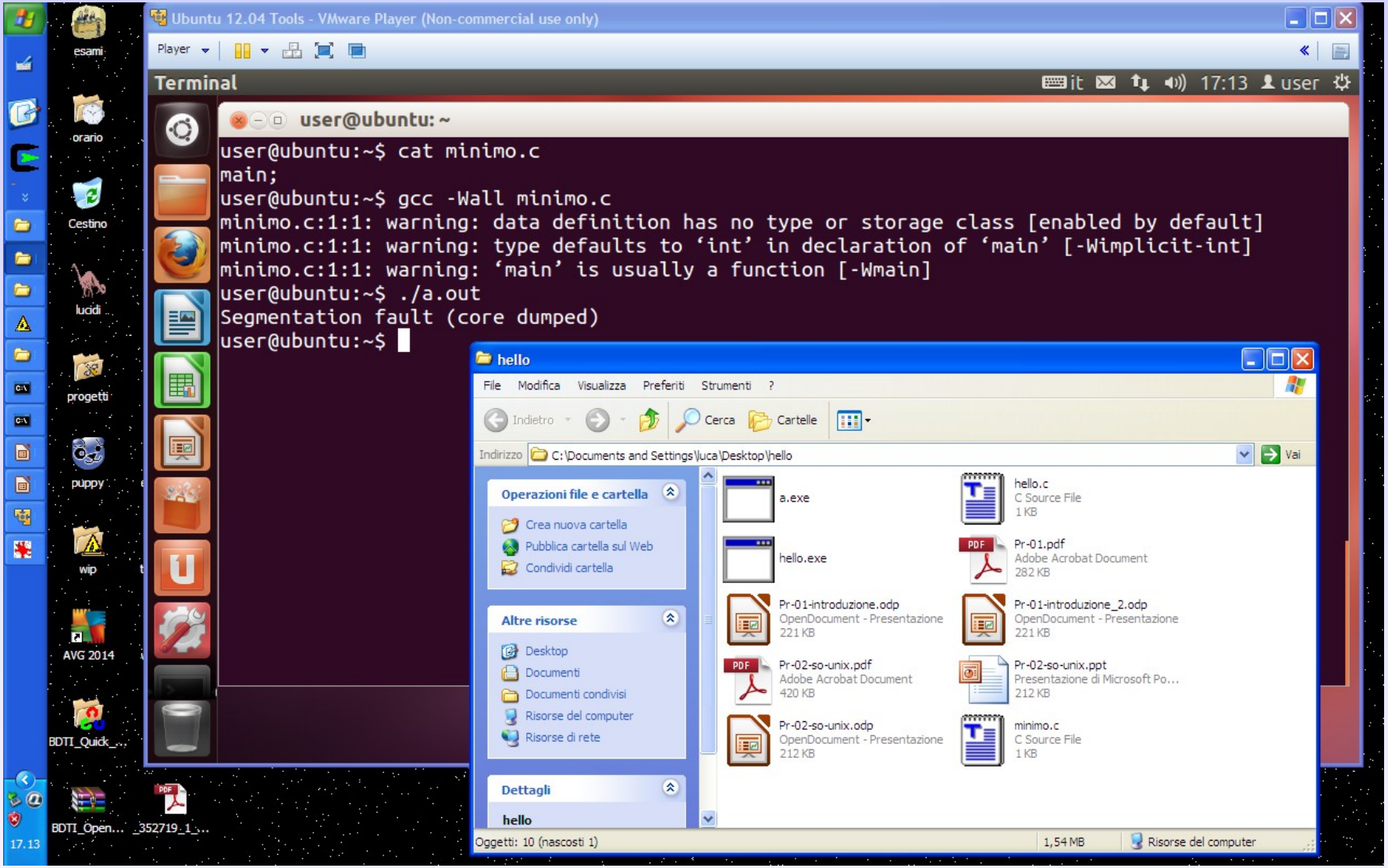
- Ha sovrascritto il file hello.c (l'originale è perduto per sempre!)
  - Però ora hello.c è un eseguibile

```
$ ./hello.c Mondo
Hello Mondo
```

Finestra cygwin in Windows



Macchina virtuale



I comandi

- **Come specificare i comandi?**
- **Scrivendo il loro nome completo, 2 possibilità:**
  - **Path assoluto**

    ```
    /usr/bin/ls
    ```
  - **Path relativo**

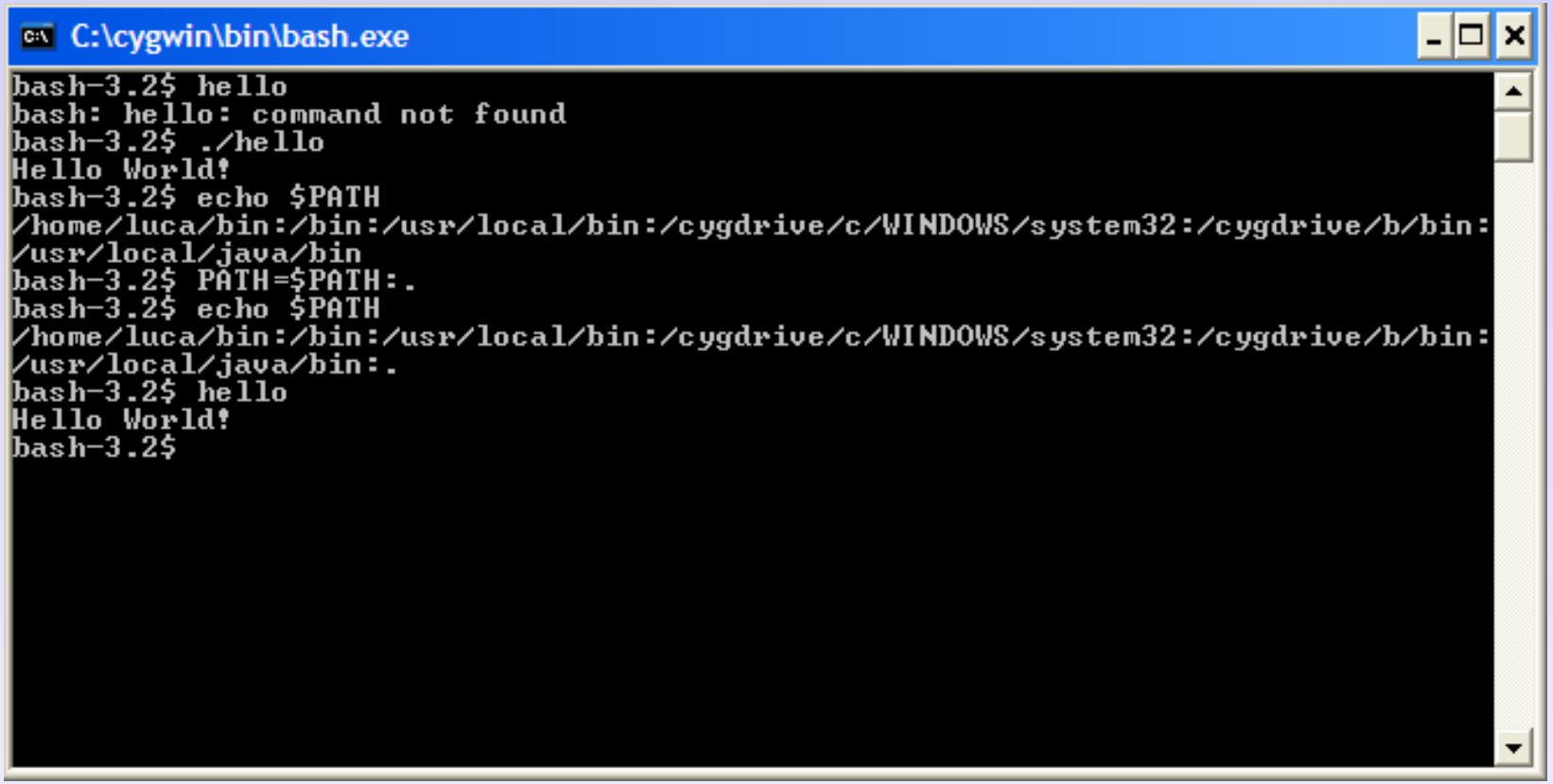
    ```
    ./a.out
    ```
- **Scrivendo solo il nome**

  ```
  ls
  ```
  - **Occorre che il comando sia in una posizione nota al sistema, in UNIX spesso `/usr/bin`**
  - **La "variabile d'ambiente" PATH elenca le directory (separate da ":") ove fare la ricerca**

    ```
    $ echo $BIN
    /usr/bin:/bin:/usr/local/bin:/home/pippo/bin
    ```
  - **Qualunque variabile d'ambiente è modificabile**

    ```
    $ <nome-variabile>=<nuovo-valore>
    ```

La variabile PATH



- Normalmente il comando creato non è direttamente eseguibile, occorre modificare la “Variabile d'ambiente” “PATH”
  - Per motivi di sicurezza si sconsiglia di includere “.” (la directory di lavoro) nel PATH (soprattutto in prima posizione)

Variabile PATH -Windows

- **In Windows il discorso è sostanzialmente analogo**
  - **Le variabili sono indicate con "%" e non "$"**

    echo %PATH%
  - **Per indicare i percorsi si usa "\" invece di "/"**
  - **I percorsi assoluti iniziano con "c:\" (in generale 'lettera':\)**
  - **Il carattere separatore nelle liste è ";" invece di ":"**
  - **La directory di lavoro è sempre inclusa implicitamente nel PATH**

Variabile PATH -Windows