Corso di

Sistemi Operativi

Classe 1 ---- Matricole congrue a 0

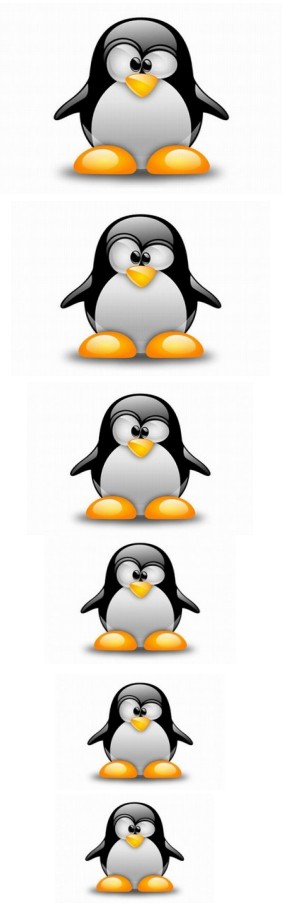
Ulteriori informazioni

- *GaPiL - Guida alla Programmazione in Linux*
  https://gapil.gnulinux.it/

- *Guida dell'utente di Linux*,
  L. Greenfield, http://www.pluto.it/files/ildp/guide/GuidaUtente/index.html (in italiano)

- *Bash Reference Manual*,
  Free Software Foundation
  http://www.gnu.org/software/bash/manual/bash.html

- *Appunti di Linux – Bash*
  http://appunti.linux.it/a2/a228.htm#almltitle777 (in italiano)



Elementi introduttivi al sistema operativo UNIX (LINUX)

Standard Unix
&
Implementazioni

Standard Unix: ANSI

- American National Standards Institute: venditori + utenti
- Membro dell' International Organization for Standardization (ISO)
- 1989: ANSI C per la standardizzazione del linguaggio C
  - portabilità di programmi C ad una grande varietà di S.O. e non solo per UNIX
  - Definisce non solo la semantica e la sintassi ma anche una libreria standard divisa in 15 aree (individuate dagli header, vedi fig. 2.1)

Standard Unix: IEEE POSIX

- Institute of Electrical and Electronic Engineers propose una famiglia di standard
  - Portable Operating System Interface
  - Lo standard 1003.1 relativo a interfacce di s.o.: definizione di servizi che un s.o. deve fornire per essere POSIX COMPLIANT
    - Definisce una interfaccia non una implementazione
    - Non è fatta distinzione tra system call e funzioni di libreria
    - Non prevede la figura di “superuser”, ma certe operazioni richiedono appropriati privilegi

Standard Unix: XPG3

- X/Open: gruppo di venditori di Computer
- Hanno prodotto 7 volumi di una guida di portabilità
- X/Open Portability Guide, Issue 3 1989
- Il vol. 2 definisce interfacce per un s.o. Unix-like, a partire da IEEE 1003.1, ma con variazioni (e.g. msg in varie lingue)

Implementazioni: SVR4

- System V Release 4 è stato prodotto dalla AT&T
- SVR4 è conforme a POSIX e XPG3

Implementazioni: 4.3+BSD

- Berkeley Software Distributions (sono distribuite da UCB)
- Conforme allo standard POSIX
- Benché fosse inizialmente legato a codice sorgente AT&T e quindi alle sue licenze, venne poi creata una versione (free) molto interessante per PC (intel based) FreeBSD

Implementazioni: Linux

- Il primo kernel sviluppato da Linus Torvalds nel 1991
- Conforme a POSIX, ma include anche la maggior parte di funzioni di SVR4 e 4.3BSD
- Disponibile su Intel, Compaq (ex Digital), Alpha, Sparc, McIntosh e Amiga
- Free

Distribuzioni Linux

- Ubuntu
- Debian
- Fedora
- RedHat
- SuSe
- Elementary OS
- Manjaro
- ...

Introduzione a Bash (Bourne Again Shell)

Che cos'è una "shell"

- Una "nicchia", cioè un ambiente comodo e protetto in cui lavorare
- Un programma che utilizza comandi in formato testo per rivolgersi al sistema operativo
- Un esempio: DOS
- **sh** (Bourne shell), **bash**, **csh** (C shell), **ksh** (Korn shell), **zsh** (Z shell), etc.
- Nel seguito si farà (quasi) sempre riferimento alla **bash**

Perché usare la shell

- Per gestire facilmente computer remoti
- Per realizzare operazioni complesse su insiemi di file
- Per ripetere le stesse operazioni su un insieme di elaboratori

Bash: Introduzione

- Sviluppata, a partire dal 1988, nel progetto GNU
- progetto GNU:
  - sviluppare una versione gratuita di UNIX
  - GNU = Gnu's Not Unix (acronimo ricorsivo)
  - open software, FSF
  - “copyleft”: software sotto “copyleft” è distribuito gratis con il codice sorgente e deve essere mantenuto tale
    - non si può vendere software preso gratis
- Steve Bourne ha scritto una delle prime shell per UNIX (1979): **sh**
- **Bash**: **B**ourne **A**gain **Sh**ell, in tributo a S. Bourne

Introduzione

- L’indipendenza della shell dal sistema operativo Unix/Linux ha portato ad una grande varietà di shell
  - sh (include reminiscenze di ALGOL)
  - csh (usa una sintassi simile al C, prende comandi interattivi o programmi)
  - tcsh (come la csh + permette di viaggiare su e giù per la lista dei comandi eseguiti, spelling correction dei comandi)
  - bash

I comandi

- Rappresentano una richiesta di servizio al SO
- Generalmente, hanno il formato

  `comando opzioni argomenti`

  - Gli argomenti indicano l'oggetto su cui eseguire il comando
  - Le opzioni (di solito) iniziano con un "–" e modificano (specializzano) l'azione del comando
- Esempi di comandi:
  - Comandi built–in della shell
  - Script
  - Codice eseguibile

Bash: introduzione

```
bash> sort –n num_telefoni > num_tel.ordinati
```

Il compito della shell:

1. separare i token del comando:
   - sort, -n, num_telefoni, >, num_tel.ordinati
2. Determinare il significato dei token
   1. sort comando da eseguire
   2. -n opzione
   3. num_telefoni argomento
   4. > operatore di ridirezione
   5. num_tel.ordinati nome del file per la ridirezione
3. Preparare il sistema in modo tale che l’output vada nel file num_tel.ordinati
4. Cercare ed eseguire il comando sort

Comandi, argomenti e opzioni

```
bash> lp -d lp1 -h file1 file2 file3
```

- linea comandi: parole (stringhe) separate da spazi o TAB
- La prima parola è il comando
- Il resto sono gli argomenti
- Gli argomenti sono spesso nomi di file, ma non necessariamente:

```
mail rescigno
```

- Una opzione è un argomento speciale che impartisce istruzioni al comando, cioè modifica il comportamento di default
- A volte un’opzione ha un proprio argomento

Comandi di shell

Comandi base della shell

Esempi

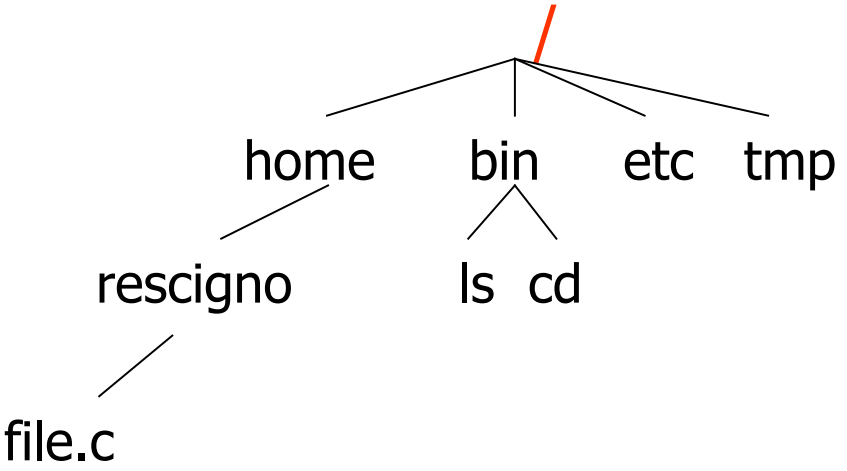
- File system: **`ls`**, **`cp`**, **`mv`**, **`rm`**, **`mkdir`**
- Accesso: **`chmod`**, **`chown`**
- Controllo del disco: **`df`**, **`mount`**
- Controllo dei processi: **`ps`**, **`kill`**, **`bg`**, **`fg`**
- Rete: **`ifconfig`**, **`ping`**, **`tracepath`**

File e directory

- Un file può contenere qualunque tipo di informazione ed esistono file di differenti tipi
  - File regolari = contengono caratteri leggibili
  - File eseguibili = sono invocati come comandi
  - Directory = sono dei contenitori in cui è possibile accedere a altri file oppure ad altre sottodirectory

File e directory

- Filesytem gerarchico
  - root /
- Nomi di file speciali:
  - dot .
  - dot dot ..



- Pathname: sequenza di zero o più nomi di file separati da /
  - pathname assoluto: pathname che descrive la posizione di un file nel sistema a partire dalla root
  - pathname relativo: pathname che descrive la posizione di un file nel sistema a partire dalla directory in cui siamo

Es: /home/rescigno/file.c

Es: rescigno/file.c

File system – Navigazione

- current working directory (cwd)
  - al login, cwd = home directory
- **pwd**: per conoscere la cwd
- **cd directory**: Change working Directory ("vai alla directory")

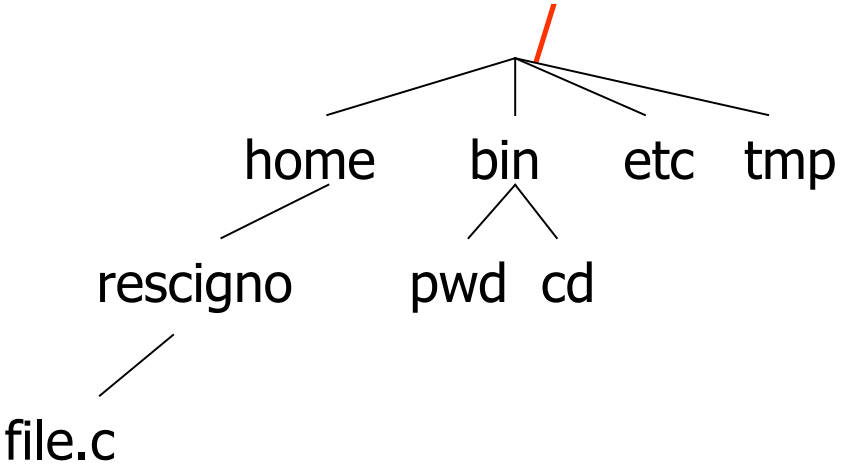
```
[/home/user/rescigno]> cd pippo
[/home/user/rescigno/pippo]> cd –
[/home/user/rescigno]> cd /usr/lib
[/usr/lib]> cd ~
[/home/user/rescigno]> cd ~giuper
[/home/user/giuper]> cd
```

File system – Navigazione

- **`mkdir directory`**: MaKe a Directory
- **`rmdir directory`**: ReMove a Directory
- **`ls directory`**: LiSt directory content
  - Esempi di opzioni:
    - **`-a`** list all files (also hidden files)
    - **`-l`** long listing
    - **`-g`** group information (not user information)
    - **`-t`** ordered by date
    - **`-R`** recursive
- **`find`**: cerca i file con un certo nome (even-tualmente definito tramite wildcard)

Directory

- Per avere l'elenco dei file presenti nella cwd : ls
  - Se si vuole conoscere l'elenco dei file presenti in una specifica directory, si passa a ls il pathname della directory




```
[/]> ls
     home  bin  etc  tmp
[/]> ls /bin
     pwd cd
[/]> cd bin
[/bin]> ls
    pwd cd
```

Attributi di un file

- Per ottenere informazioni complete su un file

```
$ ls –ls prova.txt
  1 –rw-r--r– 1 adele staff 213 Jun 2 00:12 prova.txt
```

(con **l** per lista estesa, **s** per ottenere la dimensione in blocchi)

- Si ha…
  - **1** è il numero di blocchi utilizzati dal file
  - **–rw-r--r–** sono i bit di accesso
  - **1** è il conteggio degli hard link
  - **adele** e **staff** sono rispettivamente username e groupname del possessore del file
  - **213** è la dimensione in byte
  - **Jun 2 00:12** è la data di ultima modifica
  - **prova.txt** è il nome del file
- … gran parte dei dati contenuti in un i–node

File owner

- Ogni file è associato a:
  - un utente proprietario del file
  - un gruppo (i.e. un insieme di utenti) con diritti speciali sul file
- Come identificare utenti e gruppi
  - *user_id* (valore intero) ↔ username (stringa)
  - *group_id* (valore intero) ↔ groupname (stringa)
- Come associare un utente ad un file:
  - Quando un utente crea un file, al file viene automaticamente associato il suo user_id
  - Il proprietario può "cedere il possesso" del file tramite

    `$ chown new_user_id file(s)`

Gestione dei gruppi

- Come ottenere la lista di appartenenza di un utente ai gruppi:
  `$ groups [username]`
  invocata senza argomenti, elenca i gruppi a cui appartiene l'utente che l'ha invocata, indicando **`username`** ritorna i gruppi ad esso associati
- Come associare un gruppo ad un file:
  - Quando un utente crea un file, al file viene automa-ticamente associato il suo gruppo corrente; il gruppo corrente iniziale di ogni utente è impostato dall'amministratore
  - L'utente può cambiare il proprio gruppo corrente tramite il comando
    `$ newgrp groupname`
  - Si può invece modificare il gruppo di un file con
    `$ chgrp groupname file(s)`

Permessi di accesso – 1

- Modalità relativa

  `$ chmod [ugoa] [+-] [rwxX] file(s)`

- Esempi:

  `$ chmod u+x script.sh`

  aggiunge il diritto di esecuzione per il proprietario del file **`script.sh`**

  `$ chmod -R ug+rwX src/*`

  aggiunge (ricorsivamente) i diritti di lettura e scrittu-ra per il proprietario ed il gruppo relativamente ai file contenuti nella directory **`src`**; aggiunge inoltre il diritto di esecuzione per le directory

Permessi di accesso – 2

- Modalità assoluta

User			Group			Others		
R 4	W 2	X 1	R 4	W 2	X 1	R 4	W 2	X 1

- Esempi:

  `$ chmod 755 pippo.sh`

  assegna diritto di lettura, scrittura ed esecuzione all'utente proprietario, diritto di lettura ed esecuzione al gruppo ed agli altri utenti

  `$ chmod 644 pluto.txt`

  assegna diritto di lettura e scrittura all'utente proprietario ed il solo diritto di lettura al gruppo ed agli altri utenti

Gestione dei file

- **`rm`**: ReMove (delete) files
  - Esempi di opzioni:
    - **`–r`** remove recursively beforehand
    - **`–i`** remove after confirmation
    - **`–f`** "forced" removing
    - **`$ rm –rf`**
      eseguito dal superutente sulla root, "spiana" il sistema
- **`cp`**: CoPy files
- **`mv`**: MoVe (or rename) files
- **`ln`**: LiNk creation (symbolic or not)
- **`more, less`**: page through a text file
- **`df`**: Disk Free (mostra lo spazio libero su disco)
- **`du`**: Disk Usage (mostra lo spazio utilizzato per i file contenuti all'interno di una directory)
- **`mount`**: mostra/cambia l'aspetto del grafo delle directory, montando nuovi file system (remoti)

Comandi per "mettere tutto insieme"

- **`cat`**: concatena file e ne mostra il contenuto
  - Esempio

    `$ cat file1 file2 > file3`
- **`grep`**: cerca, in un insieme di file, quelli che contengono una stringa data e riporta la linea contenente la stringa
  - Esempio

    `$ grep -r "zucchero" /home/listespesa/`
- **`sort`**: ordina le linee di un file di testo
- **`head`**, **`tail`**: mostra le prime/ultime n linee di un file

Metacaratteri – 1

- La shell riconosce alcuni caratteri speciali, chiamati metacaratteri, che possono comparire nei comandi
- Quando l'utente invia un comando, la shell lo scandisce alla ricerca di eventuali metacaratteri, che elabora in modo speciale
- Il comando viene eseguito solo dopo l'interpretazione dei metacaratteri

Metacaratteri – 2

- Esempio:

```
$ ls *.java
Albero.java div.java     ProvaAlbero.java
AreaTriangolo.java  EasyIn.java  ProvaAlbero1.java
AreaTriangolo1.java IntQueue.java
```

- Il metacarattere "*****" all'interno di un pathname è un'abbreviazione per un nome di file
- ***.java** viene espanso dalla shell con tutti i nomi di file che terminano con l'estensione **.java**
- Il comando **ls** fornisce quindi la lista di tutti e soli i file con tale estensione

Abbreviazione del pathname – 1

- I seguenti metacaratteri, chiamati wildcard, sono usati per abbreviare il nome di un file in un pathname:
  - ***** stringa di 0 o più caratteri
  - **?** singolo carattere
  - **[ ]** singolo carattere tra quelli elencati
  - **{ }** stringa tra quelle elencate

Abbreviazione del pathname – 2

- Esempi

```
$ ls /dev/tty?

/dev/ttya /dev/ttyb


$ ls /dev/tty?[234]

/dev/ttyp2 /dev/ttyp3 /dev/ttyp4

/dev/ttyq2 /dev/ttyq3 /dev/ttyq4

/dev/ttyr2 /dev/ttyr3 /dev/ttyr4
```

Abbreviazione del pathname – 3

- Esempi

```
$ ls /dev/tty?[2-4]

/dev/ttyp2 /dev/ttyp3 /dev/ttyp4

/dev/ttyq2 /dev/ttyq3 /dev/ttyq4

/dev/ttyr2 /dev/ttyr3 /dev/ttyr4
```

```
$ mkdir /user/studenti/rossi/{bin,doc,lib}
```

crea le directory **bin, doc, lib**

Il "quoting" – 1

- Il meccanismo del *quoting* è utilizzato per inibire l'effetto dei metacaratteri
- I metacaratteri a cui è applicato il quoting perdono il loro significato speciale e la shell li interpreta come caratteri ordinari

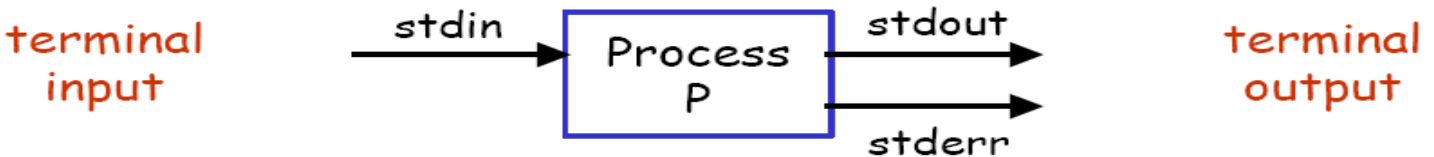
Il "quoting" – 2

- Ci sono tre meccanismi di quoting:
  - il metacarattere di **escape** **** inibisce l'effetto speciale del metacarattere che lo segue
    - Esempio

      ```
      $ cp file file\?
      $ ls file*
      file file?
      ```
  - tutti i metacaratteri presenti in una stringa racchiusa tra singoli apici perdono l'effetto speciale
  - i metacaratteri per l'abbreviazione del pathname presenti in una stringa racchiusa tra doppi apici perdono l'effetto speciale

Redirezione dell'I/O e pipe

- Ogni processo è associato a tre *stream*
  - standard input (**`stdin`**)
  - standard output (**`stdout`**)
  - standard error (**`stderr`**)
- Redirezione dell'I/O e pipe permettono di:
  - "slegare" gli stream dalle loro destinazioni abituali
  - "legarli" ad altre sorgenti/destinazioni

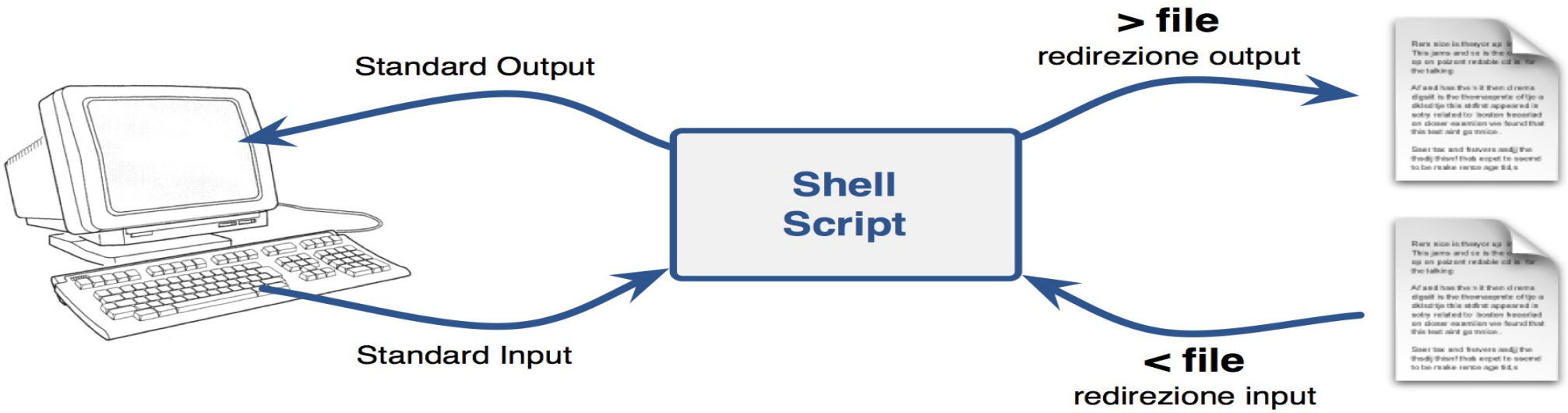


Redirezione dell'I/O – 1

- Redirezione
  - Salvare l'output di un processo su un file (output redirection)
  - Usare il contenuto di un file come input di un processo

Metacarattere	Significato
>	Redirezione dell'output
>>	Redirezione dell'output (append)
<	Redirezione dell'input
<<	Redirezione dell'input dalla linea di comando
2>	Redirezione dei messaggi di errore (bash Linux)

Redirezione dell'I/O – 2



Redirezione dell'I/O – 3

- Esempi
  - Salva l'output di **`ls`** in **`list.txt`**

    `$ ls > list.txt`
  - Aggiunge l'output di **`ls`** a **`list.txt`**

    `$ ls >> list.txt`
  - Spedisce il contenuto del file **`list.txt`** all'indirizzo arescigno@di.unisia.it

    `$ mail arescigno@di.unisa.it < list.txt`
  - Redireziona **`stdout`** del comando **`rm`** al file **`/dev/null`**

    `$ rm *.dat > /dev/null`

Redirezione dell'I/O – 4

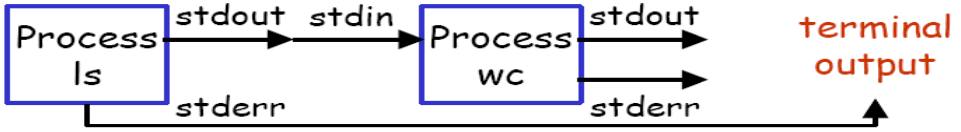
- Altri esempi

```
$ echo ciao a tutti >file #redirezione dell'output

$ more file

ciao a tutti


$ echo ciao a tutti >>file #redirezione (append)

$ more file

ciao a tutti

ciao a tutti


$ wc <progetto.txt

21 42 77
```

La pipe – 1

- La pipe, realizzata per mezzo del metacarattere |, è una catena di montaggio, serve cioè 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
- Esempio

```
$ ls
a b c d f1 f2
$ ls | wc –w
6
```



Linux

La pipe – 2

- La sequenza di comandi

```
$ ls /usr/bin > temp

$ wc -w temp

459
```

  ha lo stesso effetto della pipeline:

```
$ ls /usr/bin | wc -w

459
```

- I comandi **`ls`** e **`wc`** sono eseguiti in parallelo: l'output di **`ls`** è letto da **`wc`** mentre viene prodotto

Gestione dei processi – 1

- Attributi associati ai processi
  - pid: identificatore del processo
  - ppid: identificatore del processo padre
  - nice number: priorità statica del processo, che può essere modificata invocando il comando **`nice`**
  - TTY: terminale associato al processo
  - Effective user_id/group_id: identificatore del proprietario e del gruppo cui appartiene il proprietario
  - Memoria e cpu utilizzate
  - …

Gestione dei processi – 2

- **`ps`**: Process Status (riporta lo stato dei processi attivi nel sistema)
  - Esempio:

```
$ ps
PID   TTY      TIME       CMD
7431  pts/0    00:00:00   su
7434  pts/0    00:00:00   bash
18585 pts/0    00:00:00   ps
```

- **`nice`**: esegue un comando con una priorità statica diversa
  - Una **“`niceness`”** pari a –20 fornisce priorità massima, mentre il valore 20 corrisponde alla priorità più bassa
  - Il valore di default, per ogni processo, viene ereditato dal padre, ed è solitamente pari a 0
- **`kill`**: termina un processo

Gestione dei processi – 3

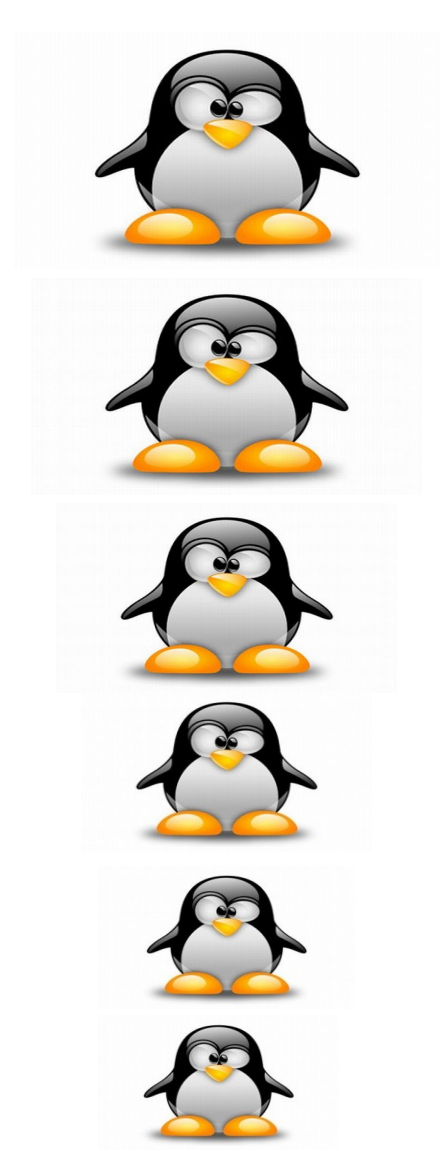
- Processi in foreground
  - Processi che "controllano" il terminale dal quale sono stati lanciati
  - In ogni istante, un solo processo è in fore-ground
- Processi in background
  - Vengono eseguiti senza che abbiano il controllo del terminale a cui sono "attaccati"
- Job control
  - Permette di portare i processi da background a foreground e viceversa

Gestione dei processi – 4

- **&**: lancia un processo direttamente in background
  - Esempio:

    $ prova &
- **^Z**: sospende un processo in foreground
- **^C**: termina un processo in foreground
- **fg**: porta in foreground un processo sospeso
- **bg**: riprende l'esecuzione in background di un pro-cesso sospeso
- **jobs**: produce una lista dei processi in background
- **n**: identifica il numero di un processo
  - Esempio:

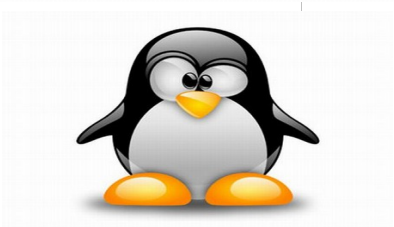
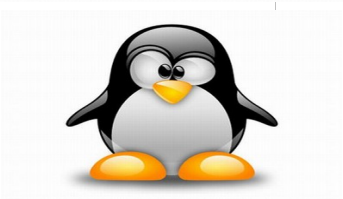
    $ kill -9 6152
- Nota: **jobs** si applica ai processi attualmente in esecuzione nella shell, mentre la lista completa dei processi in esecuzione nel sistema può essere ottenuta con i comandi **top** o **ps –ax**



Come sapere chi fa che cosa?

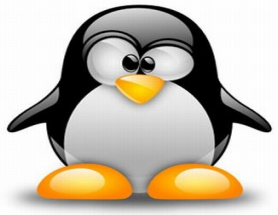
- **man**: seguito dal nome del comando, invoca il manuale in linea
- **apropos**: aiuta a reperire il comando "giusto" per eseguire una data operazione (mostrando tutti i comandi correlati ad una certa opera-zione)
- tldp.org: The Linux Documentation Project
- Google

Linux


La shell

Linux™



Selezionare una shell

Quando viene fornito un account su un sistema Linux viene anche selezionata una shell di default

Per vedere che shell si sta utilizzando:

`$ echo $SHELL`

mostra il contenuto della variabile **`SHELL`**

Per cambiare shell:

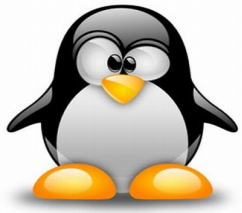
`$ cat /etc/shells` →visualizza le shell disponibili

`$ chsh` → Change Shell, visualizza fra parentesi la shell in uso; occorre specificare il percorso completo della nuova shell (es. /bin/csh)

Esempio

```
$ echo $SHELL
  /bin/bash
$ csh
  New shell [/bin/bash]: /bin/csh
```

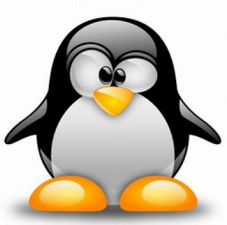
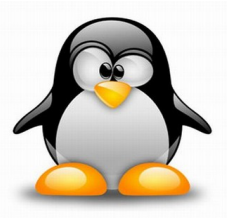
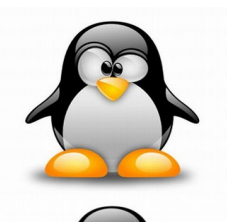
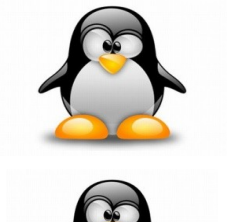
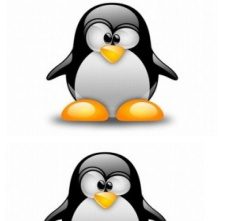
Linux



Comandi interni ed esterni – 1

- La shell riesce ad interpretare un insieme di **comandi interni** (internal command o built–in command), implementati al suo interno
- L'elenco di tali comandi è ottenibile tramite il comando interno **`help`**
- I comandi interni, assieme ai nomi dei file eseguibili ed ai simboli che rappresentano gli operatori aritmetico–logici, formano lo *scripting language* interpretato dalla shell, con cui è possibile scrivere file di testo, detti **script**, contenenti comandi che possono essere interpretati ed eseguiti dalla shell stessa
- Viceversa, quando si richiede l'esecuzione di un **comando esterno**, viene prima ricercato il corrispondente file eseguibile, che viene successivamente caricato in memoria ed eseguito

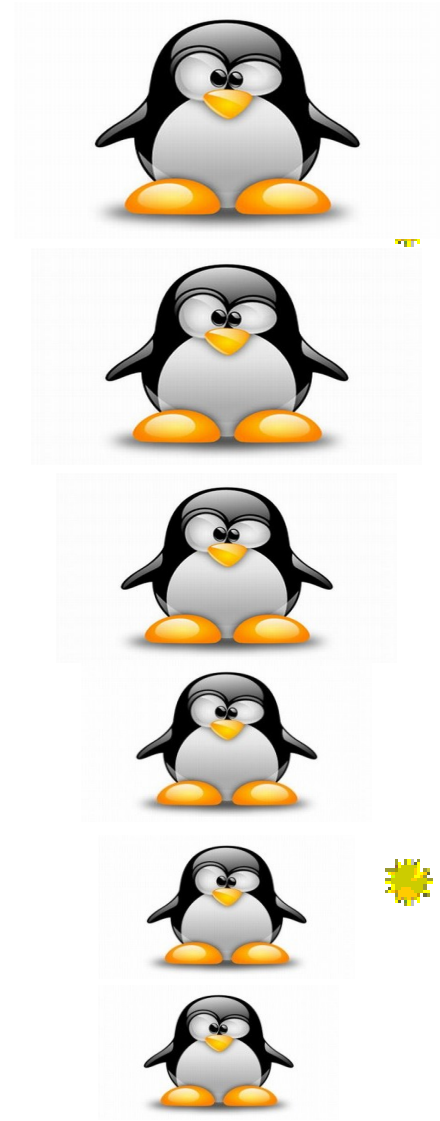
Linux

Comandi interni ed esterni – 2

- Più in dettaglio… quando viene impartito un comando, la shell "prova" ad eseguirlo effettuando, nell'ordine, i seguenti passi:
  1) Tenta di eseguire il comando interno con il nome specificato
  2) Ricerca il file specificato; se per tale file viene descritto il path (sia esso assoluto o relativo), il file viene ricercato soltanto seguendo il path; se il path non è specificato, il file viene ricercato nell'elenco delle directory contenuto nella variabile d'ambiente **PATH**
  3) Tenta di eseguire il file specificato

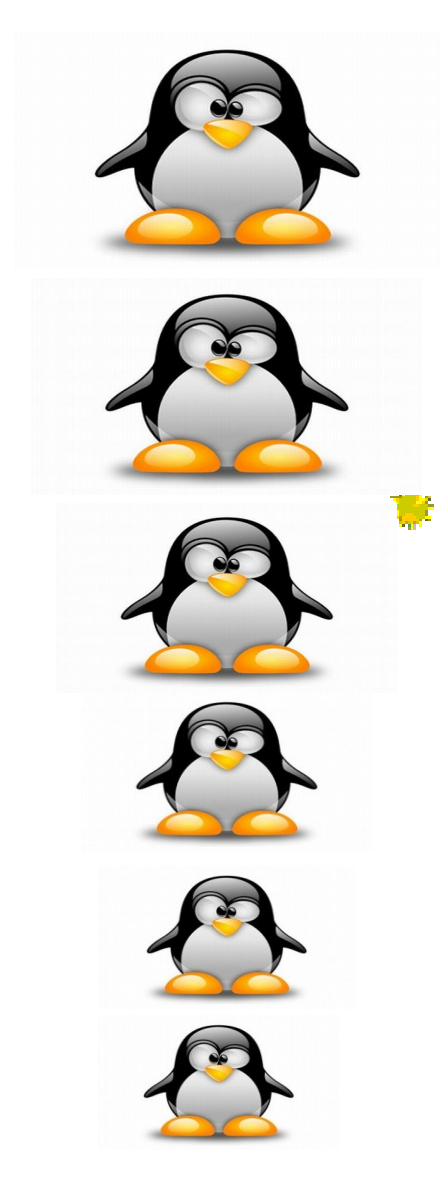
Linux



Comandi interni ed esterni – 3

- Per poter essere eseguito, un file deve esistere ed avere il permesso di esecuzione per l'utente che ha impartito il comando: in tal caso la shell crea un processo figlio per l'esecuzione delle istruzioni contenute nel file
  - Se si tratta di un file eseguibile in formato binario (ELF), la shell crea un processo figlio che esegue i comandi contenuti nel file
  - Se si tratta di un file non binario, la shell lo considera uno script
- La shell può essere terminata con il comando interno **`exit`**

Linux



Sequenze condizionali (e non)

Sequenze non condizionali

- Il metacarattere "**;**" viene utilizzato per eseguire due o più comandi in sequenza
- Esempio

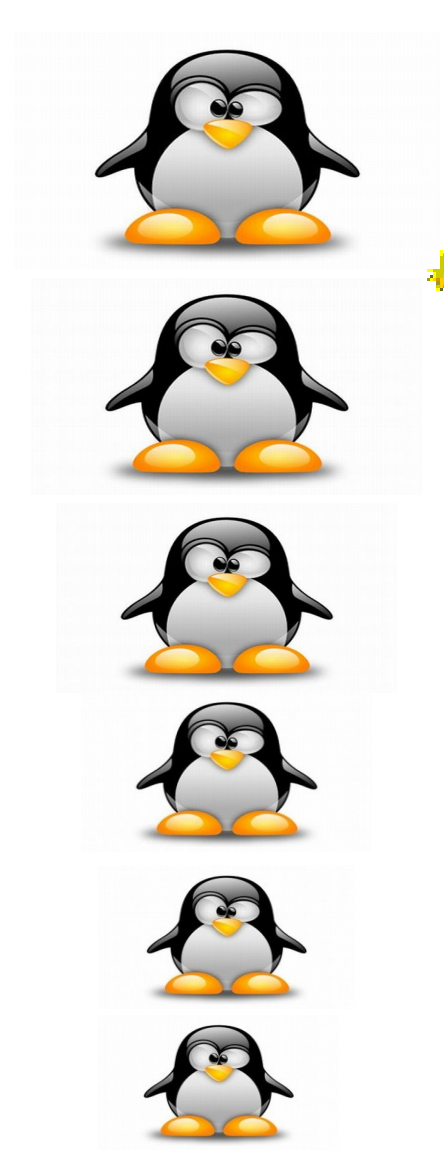
```
$ date ; pwd ; ls
```

Sequenze condizionali

- "**||**" viene utilizzato per eseguire due comandi in sequenza, solo se il primo termina con un exit code uguale ad 1 (failure)
- "**&&**" viene utilizzato per eseguire due comandi in sequenza, solo se il primo termina con un exit code uguale a 0 (success)
- Esempi

```
$ gcc prog.c –o prog && prog
$ gcc prog.c || echo Compilazione fallita
```

Linux

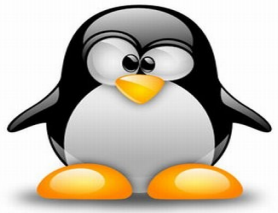


Esecuzione in background

- Se un comando è seguito dal metacarattere **&**:
  - Viene creata una subshell
  - Il comando viene eseguito in background, in concorrenza con la shell attualmente in uso
  - Non prende il controllo della tastiera
  - Utile per eseguire attività lunghe che non necessitano di interazione con l'utente
  - Esempio

```
$ find / –name passwd –print > result.txt &
$ …
$ more result.txt
/etc/passwd
```

Linux

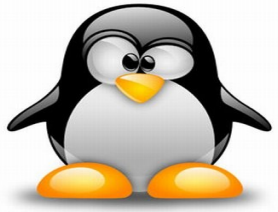


Subshell

- Quando si apre un terminale viene eseguita una shell
- Viceversa, viene creata una subshell:
  - Nel caso di comandi raggruppati
  - Quando viene eseguito uno script
  - Quando si esegue un processo in background
- Caratteristiche delle subshell
  - Hanno la propria directory corrente
  - Esempio

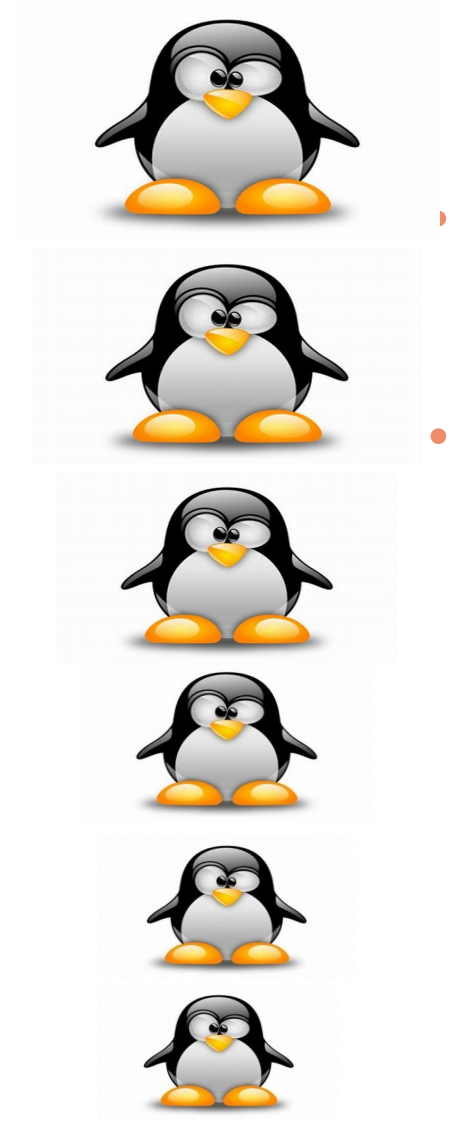
    ```
    $ cd / ; pwd
    ```
  - Vengono gestite indipendentemente due aree di variabili distinte

Linux



Variabili – 1

- Ogni shell supporta due tipi di variabili
  - **Variabili locali**, non "trasmesse" da una shell alle subshell da essa create
    - Utilizzate per computazioni locali all'interno di uno script
  - **Variabili di ambiente**, "trasmesse" dalla shell alle subshell
    - Utilizzate per la comunicazione fra parent e child shell
- Entrambi i tipi di variabile contengono stringhe
- Ogni shell ha alcune variabili di ambiente inizializzate da file di startup o dalla shell stessa
  - **$HOME**, **$PATH**, **$USER**, **$SHELL**, **$TERM**, etc.
  - Per visualizzare l'elenco completo, utilizzare il comando **env** (ENVironment)



Variabili – 2

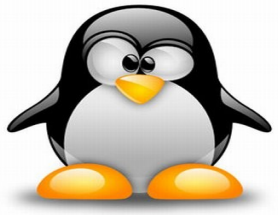
- Per accedere al contenuto di una variabile
  - Utilizzare il metacarattere **$**
  - **$name** è la versione abbreviata di **$(name)**
- Per assegnare un valore ad una variabile
  - Sintassi diversa in base alla shell
  - Nel caso di **bash**

    $ nome=valore

    $ nome="valore"
  - Variabili così dichiarate sono locali
  - Per trasformare una variabile locale in una variabile di ambiente, usare il comando **export**
  - Nel caso di **csh**

    $ setenv nome valore

Linux



Variabili – 3

- Uso di variabili locali e d'ambiente
- Esempi

```
$ firstname="monica"
$ lastname="bianchini"
$ echo $firstname $lastname
$ export firstname
$ bash
$ echo $firstname $lastname
$ exit
$ echo $firstname $lastname
```

Linux

Montare chiavetta usb

- Collegare la chiavetta usb
- Digitare il comando dmesg
  - Le ultime 2 righe indicheranno il nome del device assegnato dal sistema alla chiavetta
  - [1786.271385] sbd: sdb1
  - [1786-271391] sd 2:0:0:0: [sdb] Attached SCSI removable disk
  - In questo esempio il device è sdb1
- Creiamo una directory sotto la quale vogliamo montare la chiave usb
  - mkdir ~/usbkey
- Montiamo la chiave usb
  - sudo mount /dev/sdb1 ~/usbkey
- Per smontare
  - sudo umount /dev/sdb1

Inserire nuovo utente

- Si può procedere andando in Impostazioni, cliccando poi si Account utente; come utente amministratore sbloccare (in alto a destra) e cliccare su + in basso a sinistra. Potete così inserire nome e password del nuovo utente
  - Si crei l'utente linux (passwd: linux1210)

- Il sistema assegna automaticamente un gruppo proprio a ciascun utente, ma è possibile cambiare il gruppo di un utente usando il comando vigr che consente di accedere al file /etc/group contenete tante righe quanti sono gli utenti e per ciascuno di essi mostra il nome ma anche il group id

  user:x:1000:

  linux:x:1001:

  - 1000 nella riga di user e 1001 nella riga di linux indicano i rispettivi gruppi

Modificare il gruppo

- Per modificare il group di linux e farlo diventare uguale a quello di user si può trasformare 1001 in 1000, oppure aggiungere la parola user alla fine della riga di linux
- Uscire digitando esc e poi :wq
- Si consigli di ripetere l’operazione con il comando vigr –s che modifica il file /etc/gshadow che deve essere identico al file /etc/group (in questo caso aggiungere la parola user alla fine della linea di linux)
- Uscire digitando esc e poi :wq