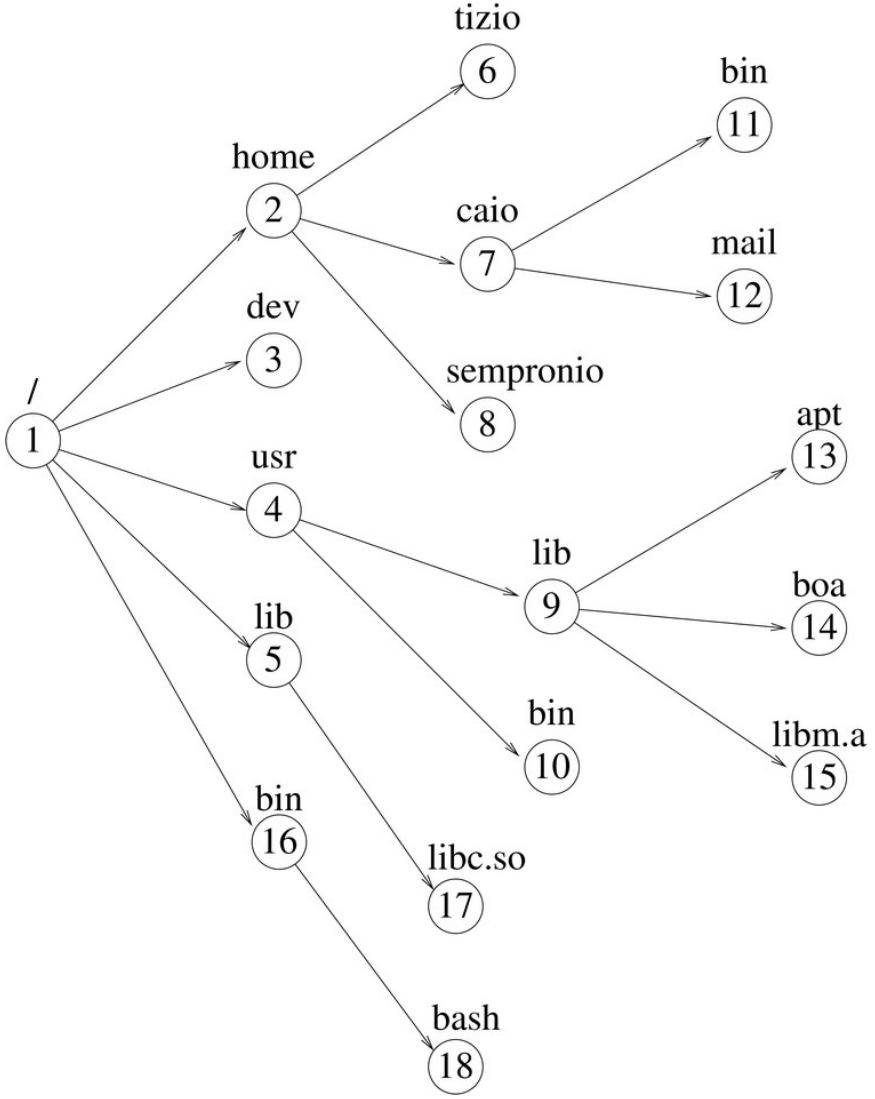
# Informazioni utili

Per gli approfondimenti sui vari punti affrontati oggi verrete indirizzati alle pagine degli “Appunti di informatica libera” ospitati nel sito:

http://appunti2.net/

# Filesystem di Unix




*I nomi di alcune directory hanno un significato ben preciso*

- /home: cartelle utenti
- /etc: configurazioni

**Cosa apprenderemo?**

- *Navigare nelle directory*
- *Creare, rinominare, copiare e cancellare files e directory*
- *Gestire i permessi*

# Navigare nel filesystem

- cd: current directory → cambia la directory
  - cd /var/www
  - cd ../lib

  *“ . e .. sono directory speciali e identificano la directory corrente e la directory padre”*

- pwd: print work directory → mostra la directory corrente
  - pwd

# Navigare nel filesystem

- ls: list → mostra i files in una directory
  - ls
  - ls /usr/sbin
- mkdir: make dir → crea una directory
  - mkdir prova
- cp: copy → copia uno o più files/directory
  - cp sorgente destinazione

    *“se destinazione è una directory, il file verrà copiato con lo stesso nome in essa.”*

# Opzioni e manuale

Ogni comando può accettare delle opzioni e in genere è fornito di un aiuto in linea e di una pagina di manuale

copia di una directory:
cp **-r** /home/utente1/backup /var/backup/users/utente1

cp –**help**

**man** cp

Spesso le pagine man di un programma/comando sono ricche di esempi e illustrano in dettaglio il funzionamento.

# Navigare nel filesystem

- mv: move → sposta/rinomina un file o una directory
  - mv file nuovonome
  - mv file directory
  - mv directory directory
- rm: remove → cancella un file o una directory
  - rm file
  - rm -rf directory

*“Molti programmi accettano una lista di files e directory”*

cp -r file1 file2 directory3 directory

mv directory1 file2 directory

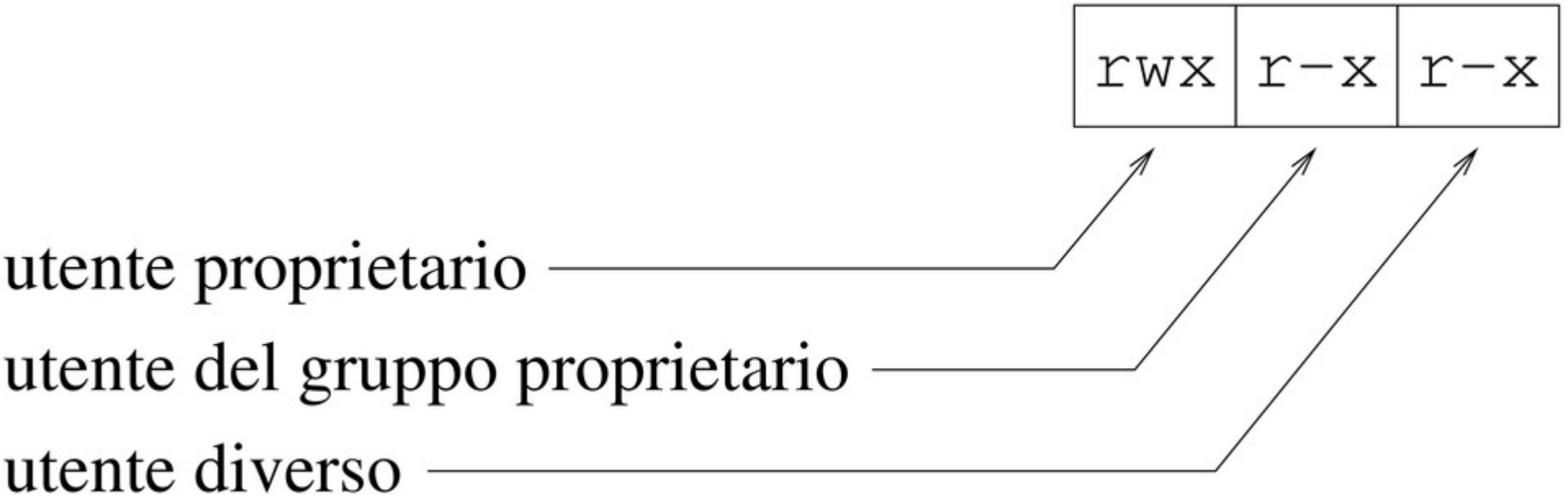
# Navigare nel filesystem

Per agevolare le operazioni su gruppi di files è possibile utilizzare i wildcards

- '*':
  - mv prova* directory → tutti i files che iniziano per 'prova'
  - mv *.txt directory → tutti i files che finiscono per '.txt'
- '?':
  - mv prova?.txt directory → es: prova**1**.txt prova**a**.txt
- '[caratteri]':
  - mv prova[234].txt directory → prova**2**.txt prova**3**.txt prova**4**.txt

# Permessi

```
$ ls -l
drwxr-xr-x 11 kbyte kbyte    4096 16 gen  2008 immagini
drwxr-xr-x 12 kbyte kbyte    4096 16 ago  2009 informatica
drwxr-xr-x  2 kbyte kbyte    4096  4 nov  2007 kaffeine
```



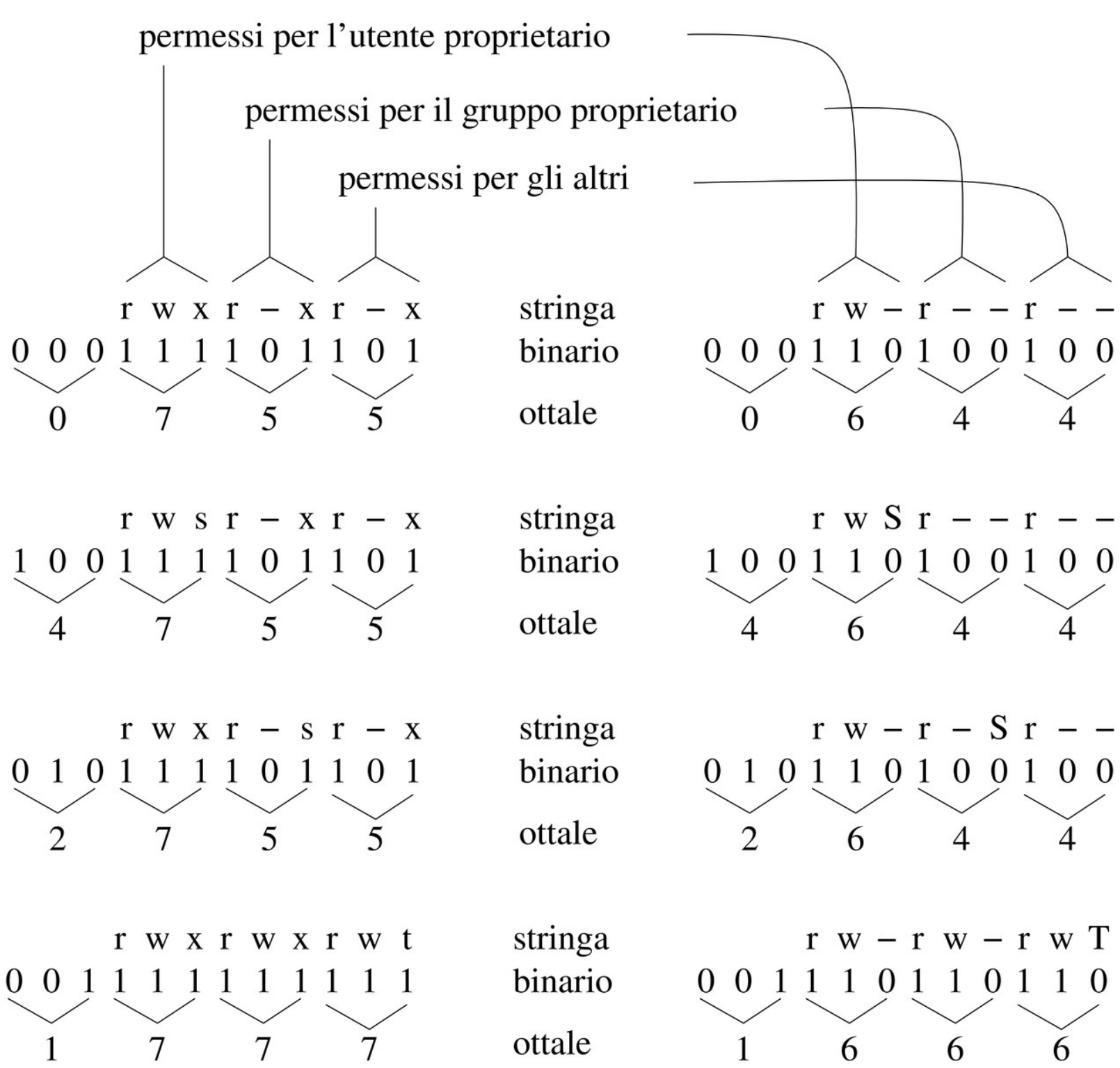
# Permessi

- chown: change owner → cambia utente proprietario
  - chown utente directory
  - chown utente:gruppo directory
  - chown -R utente directory

- chgrp: change group → cambia gruppo proprietario
  - chgrp gruppo directory
  - chgrp -R gruppo directory

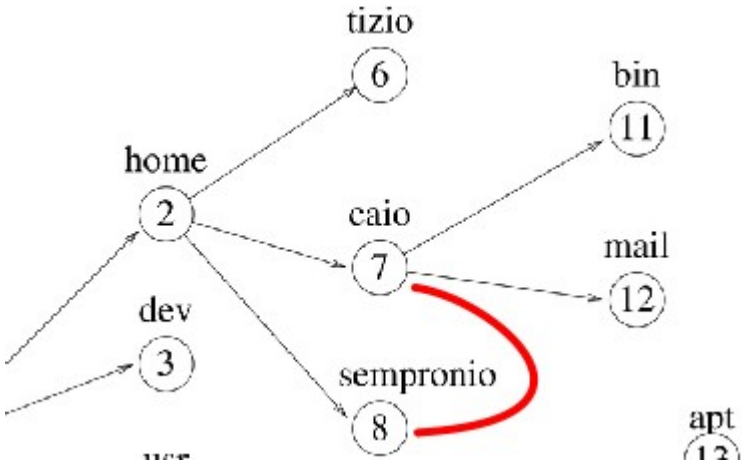
- chmod: change mode → cambia permessi lettura, scrittura esecuzione
  - chmod g-w file → toglie al gruppo proprietario il permesso di scrittura
  - chmod 755 file → imposta i permessi rwx r-x r-x

*“A volte per cambiare proprietari e permessi bisogna essere amministratori”*


permessi per l'utente proprietario
permessi per il gruppo proprietario
permessi per gli altri
r w x r – x r – x
stringa
0 0 0 1 1 1 1 0 1 1 0 1
binario
0 7 5 5
ottale
r w – r – – r – –
0 0 0 1 1 0 1 0 0 1 0 0
0 6 4 4
r w s r – x r – x
1 0 0 1 1 1 1 0 1 1 0 1
4 7 5 5
r w S r – – r – –
1 0 0 1 1 0 1 0 0 1 0 0
4 6 4 4
r w x r – s r – x
0 1 0 1 1 1 1 0 1 1 0 1
2 7 5 5
r w – r – S r – –
0 1 0 1 1 0 1 0 0 1 0 0
2 6 4 4
r w x r w x r w t
0 0 1 1 1 1 1 1 1 1 1 1
1 7 7 7
r w – r w – r w T
0 0 1 1 1 0 1 1 0 1 1 0
1 6 6 6

# Link simbolici

A volte ci capiterà di incontrare dei link simbolici. Un link simbolico è un riferimento logico ad un file o ad una directory.




Utilizzando ls li riconoscete per la diversa colorazione, in genere di colore celeste.

Eliminare un link simbolico non eliminerà le directory ed i files originali. Fa eccezione la modifica dei files.

# Link simbolici

- ln: link → crea un link
  - ln -s percorsooriginale nomelink
  - ln -s ../directory/file nomelink
  - ln -s /etc/directory directory
  - ln -s /var/directory → se non si specifica il secondo parametro verrà usato il nome del file o della directory originaria

  ***ATTENZIONE: non dimenticate il parametro "-s"!!! Senza questo parametro creerete un diverso tipo di link, chiamato hard link, che non ha le stesse caratteristiche del link simbolico.***

# Programmi che elaborano testo

- cat: concatenate file → concatena più files e li manda in output. In genere è utilizzato con un solo file in ingresso
  - cat fileesempio
  - cat fileesempo1 fileesempio2

- less e more → mostra a video un testo
  - A differenza di cat è possibile navigare nel testo, eseguire ricerche e altre operazioni di sola lettura.

# Programmi che elaborano testo

- head: testa → mostra le prime righe di un file
  - head file → mostra le prime 10 linee del file
  - head -n 30 file → mostra le prime 30 linee del file

- tail: coda → mostra le ultime righe di un file
  - tail file → mostra le ultime 10 linee del file
  - tail -n 30 file → mostra le ultime 30 linee del file
  - tail -f file → mostra le ultime linee e “monitora” il file per catturare le nuove linee. Utile per il log.

# Programmi che elaborano testo

- sed: editor di testo in linea di comando
  - Utile per modificare e sostituire del testo senza richiedere l'iterazione dell'utente. Generalmente utilizzato per modificare contemporaneamente un numero elevato di files
  - sed 's/andato/venuto/' prova.txt

    *“le espressioni regolari sono complesse, ma permettono di risolvere numerosi problemi”*

# Programmi che elaborano testo

- grep: programma per la ricerca di stringhe
  - Viene utilizzato sia per estrarre il testo che corrisponde alla ricerca, sia per determinare in quali files essa è contenuta

  - grep -r -i ciao directory → mostra le righe e il nome dei files presenti nella directory e sottodirectory che contengono la parola ciao.
  - grep -v -r -i ciao directory → come prima solo che il confronto è invertito.

  *“Grep ha numerosi comandi. Una breve lettura della pagina man può mostrare alcune delle sue potenzialità”*

# Un esempio avanzato

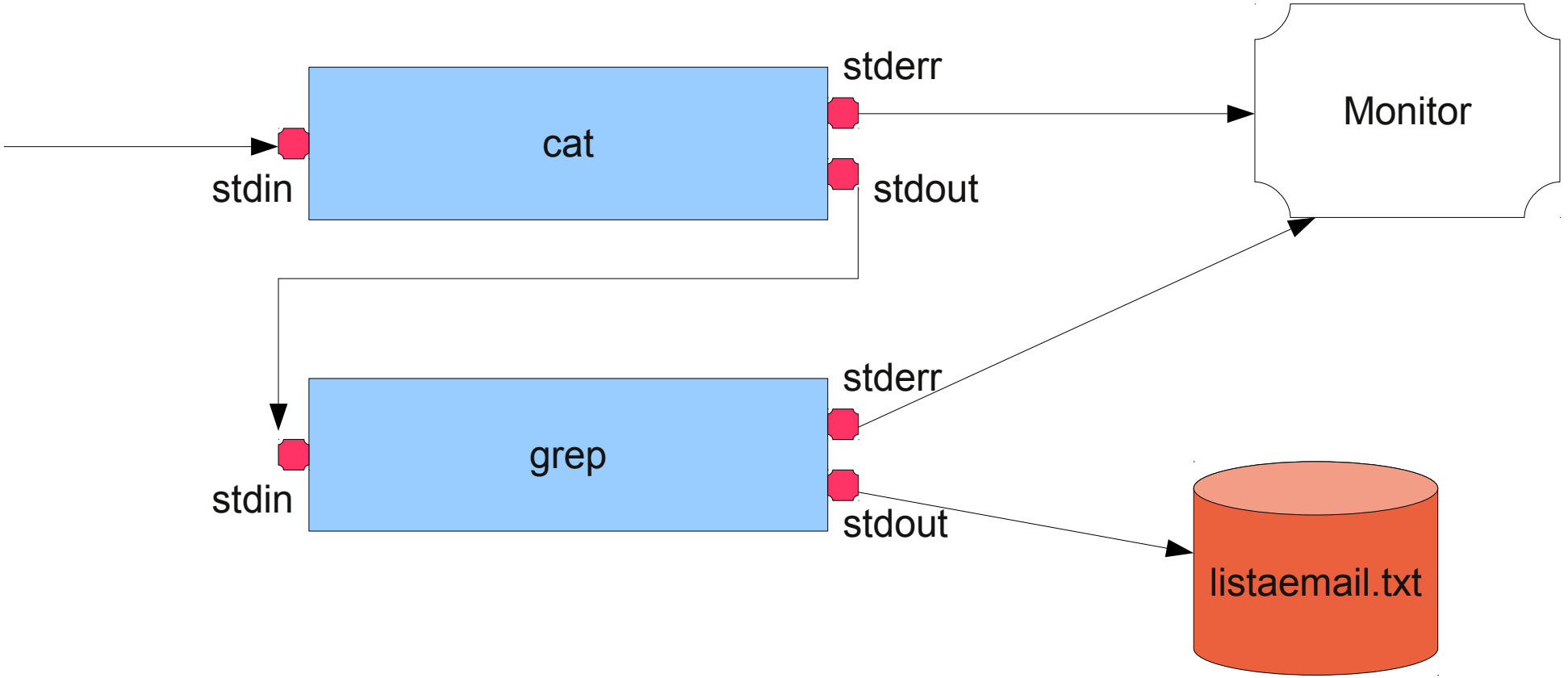
```
cat file | grep -i email > listaemail.txt
```

# Descrittori di un processo

Ogni programma che viene eseguito inizia un processo nel sistema operativo.

Ogni processo utilizza alcuni descrittori per l'input e l'output:

# Un esempio avanzato



# Pipe (Condotto)

Le pipes o in italiano “condotti” collegano lo standard output di un programma allo standard input di quello successivo.

grep email file | less
ls -l | grep rwx | tail

Ai programmi dopo un condotto non dovremo indicare loro alcun file da processare.

# Redirezioni

Le redirezioni vengono utilizzate per passare come input o utilizzare come output un file.

- Input “<”
  - grep ciao < file.txt

    questa scrittura equivale a:
  - *grep ciao file.txt* oppure *cat file.txt* | *grep ciao*

    *Tuttavia alcuni programmi non dispongono di un parametro per specificare un file in input.*

# Redirezioni

- Output “>”
  - grep ciao file.txt > risultato.txt

    Il file viene azzerato ad ogni esecuzione
  - grep ciao file.txt >> risultato.txt

    Il risultato di grep viene accodato al file

  - A differenza dei condotti, possiamo redirigere anche lo standard error
    - grep ciao file.txt > risultato.txt 2> errori.txt

# Redirezione e files speciali

Alcuni files speciali possono essere utilizzati con le redirezioni:

- /dev/null
  - Utilizzato con le redirezioni in output ignora qualsiasi dato. Utile per sopprimere l'output di un programma.
    - grep ciao file.txt 2> /dev/null
- /dev/zero e /dev/random
  - Utilizzati con le redirezioni in input forniscono una fonte inesauribile di zeri e di caratteri casuali
    - cat < /dev/random

# Approfondimenti e letture

- Appunti di informatica libera
  - 3.18 File e directory in un sistema Unix
  - 5.2.3 Utilizzo dei programmi per accedere alle pagine di manuale
  - 3.28 ABC dei comandi Unix
  - 3.24 Interpretazione dei comandi
  - 3.10 Utenti
  - 3.21 Permessi
  - 26.5 SED
  - 26.3 Grep
  - 3.25 Ridirezione e condotti

# Bash tips

- Completamento automatico
  - Premendo il tasto tab una o più volte è possibile completare automaticamente il nome di un file o di una directory.
  - In base alle estensioni installate il completamento automatico può essere di aiuto per impostare i parametri di alcuni programmi (esempio ssh).
- Ricerca tra i comandi eseguiti
  - Quando dovete eseguire nuovamente un comando svolto diverso tempo fa, ma ancora presente nella history, potete utilizzare la ricerca premendo “Control + r”.

# Problema

Vogliamo eseguire uno o più comandi su un insieme di files.

Soluzione: utilizzo di direttive sh/bash in linea di comando o creando degli script

Esempio:

for i in *.txt; do echo “Il file si chiama: “ “$i”; echo “Il suo contenuto:”; cat “$i”; done

# Necessità di uno script

Benchè molto si possa fare già da linea di comando, a volte è necessario creare degli appositi files di testo, chiamati script, che eseguono delle istruzioni bash come quella mostrata in precedenza o molto più complesse.

E' molto più semplice creare uno script che realizzare un programma in un linguaggio come java o c per svolgere gli stessi compiti.

# Editor testuale

La maggior parte degli utenti Unix/Linux utilizzano i seguenti editor testuali:

Nano

Vim

Il primo è di semplice utilizzo, mentre il secondo è molto più potente ma richiede una certa dimistichezza anche per compiti comuni quali il salvare un documento o cerare una stringa di testo nel file.

# Il nostro primo script

Prendendo spunto dall'esempio precedente, creiamo il nostro primo script:

```
#/bin/bash
for i in $1; do echo "Il file si chiama: " "$i"; echo "Il suo contenuto:"; cat "$i"; done
```

Dopo aver salvato assicuratevi di aggiungere i permessi di **esecuzione** allo script.

./script1.sh /home/kbyte/*

# Variabili predefinite

- Variabili predefinite
  - $n: parametro ennesimo passato allo script
    - Nell'esempio precedente $1 era il primo parametro, quindi “/home/kbyte/*”. I parametri sono separati da spazio. Se un parametro deve contenere spazi dovrete invocare lo script nel seguente modo:

      script.sh “parametro uno” parametrodue

  - $@: insieme di tutti i parametri

# Variabili predefinite

- $#: numero di parametri passati allo script
- $0: nome del file dello script
- $?: risultato numerico dell'ultimo comando eseguito. Questo valore viene aggiornato anche dopo l'esecuzione di un programma all'interno dello script.
- $$: numero di pid relativo all'invocazione dello script.
- $!: numero di pid relativo all'ultimo processo avviato dallo script a seguito dell'esecuzione di un programma.

# Variabili utente

E' sempre possibile definire alcune variabili all'interno di uno script:

```
VARIABILE=”ciao”
echo $VARIABILE
```

Le lettere maiuscole sono consigliate, ma non obbligatorie. Inoltre bisogna evitare di collidere con i nomi delle variabili d'ambiente.

# Variabili ambiente

Il sistema operativo e la bash definiscono alcune variabili di ambiente con lo scopo di agevolare il funzionamento di script e programmi.

L'utente loggato, la sua home, le impostazioni sul completamento automatico sono tra le informazioni memorizzate nelle variabili di ambiente.

Per un esempio di cosa contengono eseguite:

```
set | less
```

# Costrutto IF

Vogliamo aggiungere un semplice controllo sul numero di parametri passati allo script per evitare che venga invocato in modo non corretto:

```
if [ $# = 1 ]; then
  for i in $1; do echo "Il file si chiama: " "$i"; echo "Il suo
contenuto:"; cat "$i"; done
else
  echo "Numero di parametri errato"
fi
```

# Test e confronti

Le espressioni nelle parentesi quadre sono dei test il cui risultato può essere vero o falso.

Per la sintassi da utilizzare nei vari casi potete dare un'occhiata alla tabella *“19.49. Espressioni per il comando test.”* degli appunti di informatica libera o spulciando la man di “test”.

# Costrutto WHILE e UNTIL

A volte sono necessari cicli complessi che non possono essere svolti con il costrutto “for” mostrato in precedenza. In questi casi si usa while o until.

```
until lista_condizione
do
    lista_di_comandi
done

while lista_condizione
do
    lista_di_comandi
done
```

# Funzioni

Quando lo script è particolarmente complesso, può essere utile dividerlo in più funzioni:

```
#!/bin/sh
function verifica() {
    if [ -e "/var/log/packages/$1" ]
    then
        return 0
    else
        return 1
    fi
}

if verifica pippo
then
    echo "il pacchetto pippo esiste"
else
    echo "il pacchetto pippo non esiste"
fi
```

Le variabili speciali $numero assumono un significato diverso in una funzione. Non rappresentano più i parametri passati allo script, ma quelli passati alla funzione.

# Approfondimenti e letture

- Appunti di informatica libera
  - 19.1 Introduzione alla shell Unix
  - 19.2.8 Particolarità importanti della shell Bash
  - 19.3 Programmazione