Modulo 2
La shell

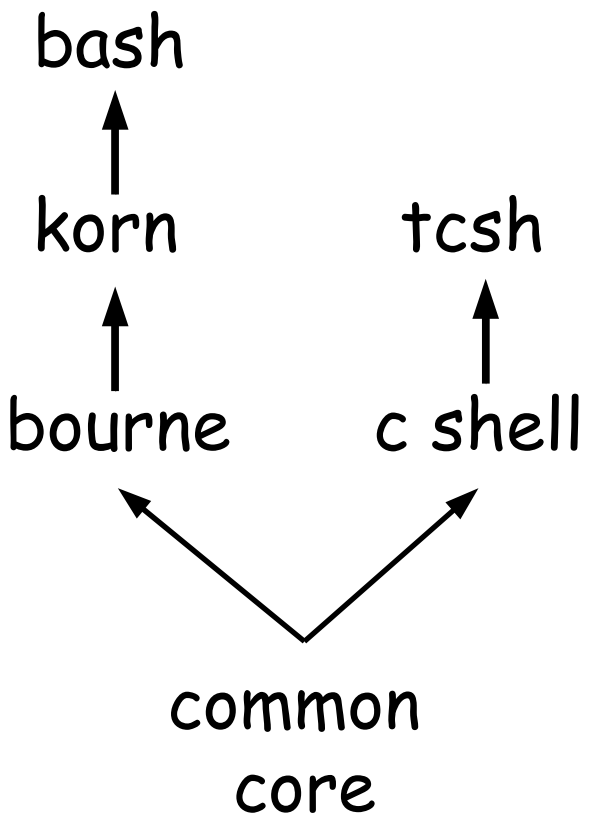
Laboratorio di Sistemi Operativi I
Anno Accademico 2008-2009

La shell di Linux

- E' un programma che si interpone fra l'utente e il SO (deve il suo nome al fatto che nasconde dietro la sua interfaccia i dettagli del SO sottostante)
- Ha la funzione di interprete della linea comandi ed esecutore di comandi su richiesta dell'utente
- Presenta all'utente il *prompt* (una stringa personalizzabile, per convenzione terminata dal carattere `%` o **$** o **>**) dopo il quale l'utente può digitare i comandi da sottomettere al sistema
- E' programmabile: permette di scrivere delle procedure in un linguaggio interpretato detto *script*
- Per analogia, le interfacce grafiche disponibili per Linux (quali KDE, GNOME) sono a volte identificate come *shell grafiche* o *visuali*

Tipi di shell

- **Esistono diversi tipi di shell**
  - Bourne shell (*sh*)
  - Korn shell (*ksh*)
  - C shell (*csh*, *tcsh*)
  - Bash (Bourne-again shell) (*bash*)
- **Quale shell scegliere?**
  - Questione di gusti ... (ma anche secondo le funzioni e sintassi offerte)
  - Bash è la più diffusa in ambiente Linux
- **Studio della shell**
  - Caratteristiche comuni (common core)
  - Analisi della bash



Caratteristiche delle Shell

- **Sommario:**
  - Comandi interni
  - Metacaratteri
  - Variabili
    - Locali
    - Di ambiente
  - Redirezione dell'I/O
  - Pipes
  - Wildcards
  - Sequenze
    - Condizionali
    - Non condizionali
  - Raggruppamento di comandi
  - Esecuzione in background
  - Command substitution
  - Quoting
  - Utili comandi esterni

Meccanismo di funzionamento

- Quando l'utente digita una linea di comando e la sottomette alla shell mediante il tasto **<RETURN>**, la shell estrae innanzitutto la prima parola
- Assumendo che sia il nome di una programma, lo ricerca nel filesystem (a meno che non sia un comando interno) e lo manda in esecuzione
- Quindi si sospende fino alla terminazione del programma, dopo la quale assume di nuovo il controllo e visualizza il prompt, in attesa del prossimo comando
- Da notare che la shell è un ordinario programma che viene eseguito nello spazio utente, che legge/scrive da/su terminale e che ha la capacità di lanciare altri programmi

Selezionare una shell

- Quando vi viene fornito un account UNIX, una shell di default viene selezionata per voi
- Per vedere che shell state utilizzando:

```
% echo $SHELL # display the content of variable
  SHELL
```

- Per cambiare shell:

```
% chsh [<username>] # asks for the full pathname of
  the new shell
```

```
[penguin@artarctic ~]$ echo $SHELL
/bin/bash
[penguin@artarctic ~]$ chsh
Changing shell for penguin.
Password:
New shell [/bin/bash]: /bin/tcsh
Shell changed.
[penguin@artarctic ~]$
```

Subshells

- **Quando aprite un terminale, viene eseguita una shell**
- **Viene creata una child shell (o subshell)**
  - Quando viene eseguito uno script
  - Quando viene eseguito un processo in background
  - Nel caso di comandi raggruppati, come **`(date; ls; pwd)`**
- **Caratteristiche delle subshell:**
  - Hanno la propria directory corrente:

```
% (cd / ; pwd)
/
% pwd
/home/penguin
```

  - Due distinte aree di variabili vengono gestite differentemente (vedi oltre)

Metacaratteri

- **Caratteri speciali**
  - `>, >>, <` redirezione I/O
  - `|` pipe
  - `*, ?, […]` wildcards
  - `` `command` `` command substitution
  - `;` esecuzione sequenziale
  - `||, &&` esecuzione condizionale
  - `(…)` raggruppamento comandi
  - `&` esecuzione in background
  - `"", ''` quoting
  - `#` commento
  - `$` espansione di variabile
  - `\` carattere di escape
  - `<<` “here documents”

Comandi

- rappresentano una richiesta di esecuzione
- sintassi generale: **`comando`** ***`opzioni argomenti`***
- gli argomenti indicano l'oggetto su cui eseguire il comando
- le opzioni generalmente:
  - iniziano con - (seguito da una lettera) o -- (seguito da una parola)
  - modificano l'azione del comando
- **Esempi di comandi:**
  - comandi interni della shell
  - script interpretati
  - programmi eseguibili

Tasti di controllo nella shell

`Ctrl-s` sospende la visualizzazione

`Ctrl-q` riattiva la visualizzazione

`Ctrl-c` interrompe il processo

`Ctrl-z` ferma il processo

`Ctrl-d` end-of-file

Comandi interni ed esterni

- **Interni, o *built-in***

  Il comando viene riconosciuto ed eseguito nel contesto (*processo*) della shell corrente.

  Esempi:

```
echo    # displays all of its arguments to standard
  output
cd      # change working directory
```

- **Esterni**

  Il comando corrisponde ad un file eseguibile che viene cercato nel filesystem, caricato in memoria ed eseguito come nuovo processo.

  Esempio: il comando `ls` si trova in `/bin/ls`

Variabili

- **Ogni shell gestisce le variabili in due modi diversi:**
  - *Variabili locali*: non ereditate dalle subshell della shell di partenza
    - Utilizzate per elaborazioni *locali* all'interno di uno script
  - *Variabili di ambiente*: ereditate dalle subshell create dalla shell
    - Utilizzate per comunicazioni da *parent* a *child* shell
- **In ogni caso le variabili contengono solo dati di tipo stringa**
- **Ogni shell ha alcune variabili di ambiente inizializzate da file di startup o dalla shell stessa**
  - **`$HOME`, `$PATH`, `$MAIL`, `$USER`, `$SHELL`, `$TERM`**, etc.
  - Per visualizzare l'elenco completo, usate il comando **`env`**

Significato di alcune variabili d'ambiente predefinite

`DISPLAY`	used by the X Window system to identify the display server
`HISTSIZE`	size of the shell history file in number of lines
`HOME`	path to your home directory
`HOSTNAME`	local host name
`LANG`	preferred language
`LD_LIBRARY_PATH`	paths to search for libraries
`LOGNAME`	login name
`MAIL`	location of your incoming mail folder
`MANPATH`	paths to search for man pages
`PATH`	search paths for commands
`PPID`	process ID of the shell's parent. This variable is readonly.
`PS1`	primary prompt
`PS2`	secondary prompt
`PWD`	present working directory
`SHELL`	current shell
`TERM`	terminal type
`UID`	user ID

Utilizzo delle variabili

- **Per accedere al contenuto di una variabile:**
  - Utilizzate il metacarattere **`$`**
  - **`$name`** è la versione abbreviata di **`${name},`** da usare solo quando non vi siano ambiguità nell'interpretazione
  - se si vuole estrarre una sottostringa: **`${name:start:len}`**
- **Per assegnare un valore ad una variabile:**
  - Sintassi diversa a seconda della shell
  - Nel caso di bash:

```
nome=valore      # problem with spaces
nome="il valore"# no problem with spaces
```

  - Variabili dichiarate in questo modo sono locali
  - Per trasformare una variabile locale in una d'ambiente, usate il comando **`export`**

```
% export nome
```

Ancora sulle sottostringhe

- E' possibile eliminare la più lunga o la più breve sottostringa iniziale:

```
% myvar="foodforthought"
% echo ${myvar##*fo}
rthought
% echo ${myvar#*fo}
odforthought
```

- E' possibile eliminare la più lunga o la più breve sottostringa finale:

```
% myvar="foodforthought"
% echo ${myvar%%fo*}

% echo ${myvar%fo*}
food
```

Esercizi

- Prevedere l'output del sistema in questa sequenza di comandi:

```
% firstname="Penguin"
% lastname="Blackwhite"
% echo $firstname $lastname
% export firstname
% bash
% echo $firstname $lastname
% firstname="Zebra"
% exit
% echo $firstname $lastname
```

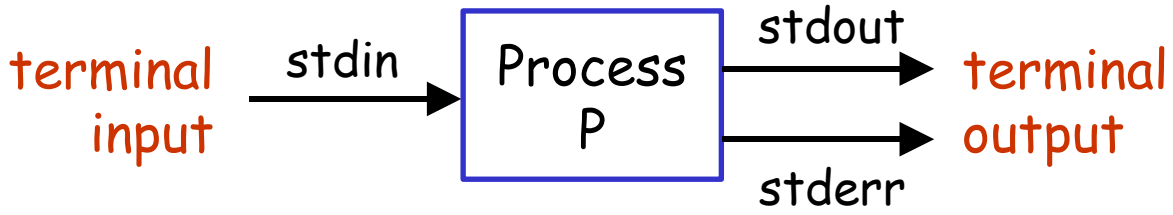
- Sia **myvar='prova1'** e si voglia creare la nuova variabile **myvar2** con valore **'abcprova1def'**: quale sintassi usare per l'assegnazione?

- Sia **myvar='unoduetre'** e si voglia creare tre variabili contenenti rispettivamente **'uno' 'due' 'tre'**: come estrarle da **myvar**?

Redirezione dell'input/output e pipes

- **Ogni processo è associato a tre *stream* (flussi di byte)**
  - Standard input (*stdin*) (rediretto con il simbolo < oppure 0<)
  - Standard output (*stdout*) (rediretto con il simbolo > oppure 1>)
  - Standard error (*stderr*) (rediretto con il simbolo 2>)
  - ***Ulteriori notazioni:***
    >& redirige *stdout* e *stderr* insieme
    x>&y file descriptor x diventa un duplicato del file descriptor y

- **Redirezione dell'I/O e pipe permettono:**
  - "Disconnettere" questi stream dalle loro sorgenti/destinazioni abituali (o di default)
  - "Connetterli" ad altri sorgenti/destinazioni



Redirezione dell'input/output (I)

- **Usi della *redirection***
  - Salvare l'output di un processo su un file (output redirection)
  - Usare il contenuto di un file come input di un processo
- **Esempi:**
  - Salva l'output di **`ls`** in **`list.txt`**

    ```
    ls > list.txt
    ```

  - Aggiunge (*append*) l'output di **`ls`** a **`list.txt`**

    ```
    ls >> list.txt
    ```

  - Spedisce il contenuto di **`list.txt`** a **`penguin@antarctic.org`**

    ```
    mail penguin@antarctic.org < list.txt
    ```

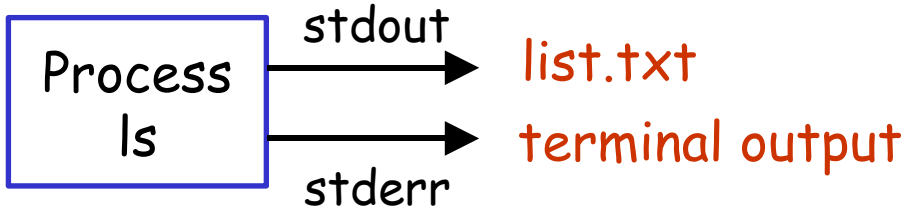
  - Redireziona **`stdout`** e **`stderr`** del comando **`rm`** al file **`/dev/null`**

    ```
    rm file >& /dev/null
    ```

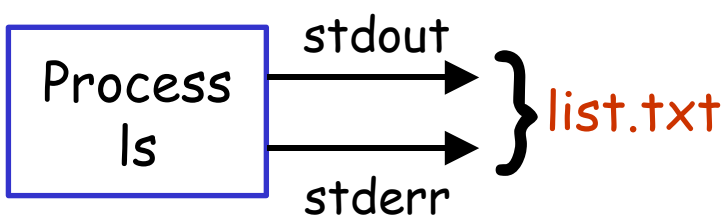
Redirezione dell'input/output (II)

```
mail penguin@antarctic.org < list.txt
```


```
ls > list.txt
```




```
ls >& list.txt
```



Redirezione dell'input/output (III)

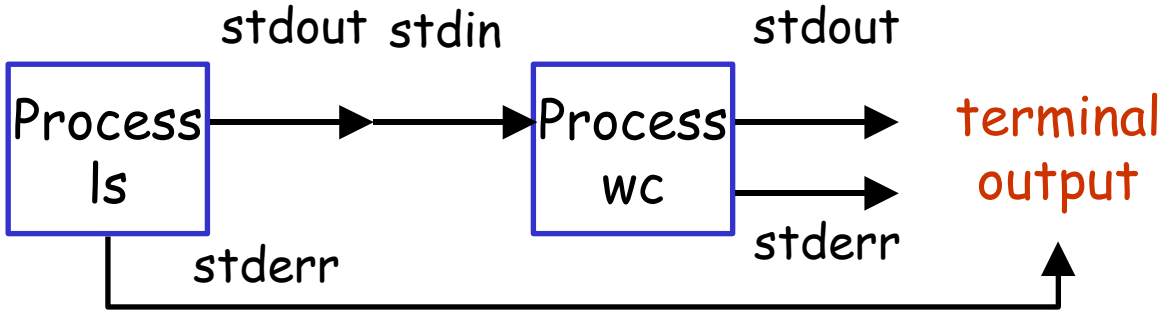
Altri esempi:

```
[penguin@artarctic ~]$ cat 1> myfile
dati del file
<Ctrl+d>
[penguin@artarctic ~]$ cat myfile
dati del file
[penguin@artarctic ~]$ cat myfile myfill > myfile2 2> myfilerr
[penguin@artarctic ~]$ cat myfile2
dati del file
[penguin@artarctic ~]$ cat myfilerr
cat: myfill: No such file or directory
[penguin@artarctic ~]$ cat myfile myfill >& myfile3
[penguin@artarctic ~]$ cat myfile3
dati del file
cat: myfill: No such file or directory
[penguin@artarctic ~]$
```

Pipes

- **Pipe, o catena di montaggio:**
  - La shell vi permette di usare lo standard output di un processo come standard input di un altro processo

```
% ls
a b c d f1 f2
% ls | wc -w
6
```



- **Esempio:**
  - **`who | cut -c1-8 | sort -u | pr -l1 -8 -w78 -t`**

`who` lista gli utenti che hanno fatto login

`cut -c1-8` mostra le colonne da 1 a 8 dell'output di **`who`**

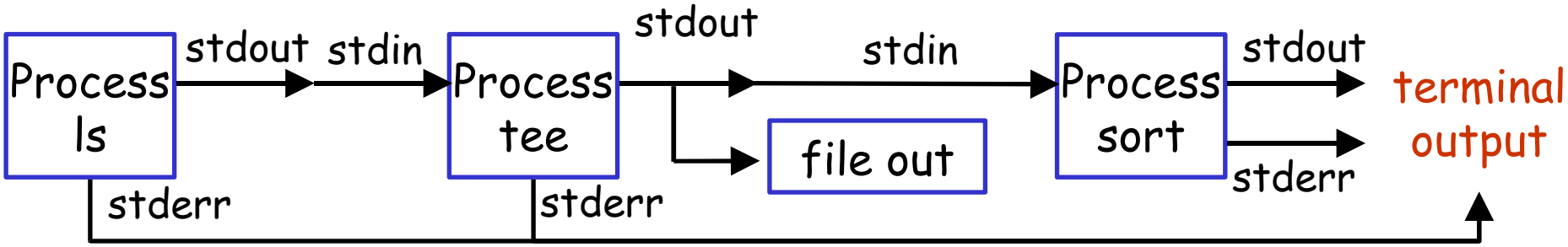
`sort -u` ordina in nomi ed elimina righe multiple con stesso nome

`pr -l1 -8 -w78 -t` mostra le informazioni disposte su 8 colonne, un nome per colonna, su linee di 78 caratteri

Duplicazione dell'output

- **Comando `tee`**
  - copia il contenuto dello standard input sia sul file specificato sia sullo standard output (il nome **`tee`** deriva dai "giunti a T" usati dagli idraulici!)
  - utile quando si vuole mandare l'output in una pipe ma allo stesso tempo salvarlo in un file

```
[penguin@artarctic]$ ls -t -1 *.txt | tee out | sort
due.txt
tre.txt
uno.txt
[penguin@artarctic]$ cat out
tre.txt
due.txt
uno.txt
```



Wildcards per filename expansion

- **Utilizzate per specificare filename pattern**
  - La shell sostituisce la stringa contenente *wildcards* con l'elenco dei file che soddisfano la condizione
  - Caratteri speciali:
    - `*` matching di qualsiasi stringa, anche vuota
    - `?` matching di qualsiasi carattere singolo
    - `[…]` matching di qualsiasi carattere inserito nelle parentesi
  - Esempi:
    - `*.c`
    - `prova00?.c`
    - `prova[0-9][!3][v,V].txt`

Esercizi

- Spiegare in dettaglio come la shell tratta il seguente comando:
  `mail penguin@antarctic.org < list.txt`
- Scrivere un comando che emuli il comando **`ls *file*`** senza usare il comando **`ls`**
- Il comando **`grep PATTERN [FILE ...]`** cerca nei file indicati l'espressione regolare **`PATTERN`** e stampa solo le linee che la contengono; esiste anche l'opzione **`-r (o -R)`** che permette di cercare **`PATTERN`** ricorsivamente nei file della directory indicata e in tutte le subdirectory ricorsivamente; scrivere un comando che emuli **`grep -r PATTERN DIR`**, utilizzando grep (ed altri comandi) ma senza ricorrere all'opzione **`-r`**
- Mandare in pipe sia lo stdout che lo stderr di un comando nell'input del comando successivo

Sequenze (condizionali e non)

- **Sequenze non condizionali**
  - Il metacarattere `;` viene utilizzato per eseguire due comandi in sequenza
  - Esempio:

```
% date ; pwd ; ls
```

- **Sequenze condizionali**
  - `||` viene utilizzato per eseguire due comandi in sequenza, solo se il primo ha un exit code uguale a **1** (*failure*)
  - `&&` viene utilizzato per eseguire due comandi in sequenza, solo se il primo ha un exit code uguale a **0** (*success*)
  - Esempi:

```
% gcc prog.c && a.out
% gcc prog.c || echo Compilazione fallita
```

Raggruppamento di comandi

- **E' possibile raggruppare comandi racchiudendoli tra parentesi**
  - Vengono eseguiti in una subshell dedicata
  - Condividono gli stessi *stdin, stdout* e *stderr*

```
[penguin@artarctic]$ date ; ls ; pwd > out.txt
Mon Sep  4 23:45:34 CEST 2006
Desktop  DUMMY  main.c   myfile2   out
[penguin@artarctic]$ cat out.txt
/home/penguin
[penguin@artarctic]$ (date ; ls ; pwd) > out.txt
[penguin@artarctic]$ cat out.txt
Mon Sep  4 23:45:34 CEST 2006
Desktop
DUMMY
main.c
myfile2
out
/home/penguin
```

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 che state utilizzando
  - non prende il controllo della tastiera
  - utile per eseguire attività lunghe che non richiedono input dall'utente
  - Esempi:

```
% find / -name passwd -print &
20123
% /etc/passwd
% find / -name passwd -print >& results.txt &
20124
```

PID del processo find

output del comando find lanciato in background

Command substitution

- **Gli apici ` ` (accento grave) sono utilizzati per fare *command substitution***
  - Il comando racchiuso fra apici viene eseguito, e il suo standard output viene sostituito al posto della stringa del comando
  - Esempi:

```
% echo Data odierna: `date`
% echo Utenti collegati: `who | wc -l`
% tar zcvf src-`date`.tgz src/
```

- **Sintassi alternativa con `$(...)`:**

```
% today=$(date)
```

Quoting

- **Esiste la possibilità di disabilitare wildcard / command substitution / variable substitution (cioè l'interpretazione della shell di questi metacaratteri)**
  - Single quotes `''` inibiscono wildcard, command substitution, variable substitution
  - Esempio:

```
% echo 3 * 4 = 12
% echo '3*4 = 12'
```

  - Double quotes `" "` inibiscono wildcard e basta
  - Esempio:

```
% echo " my name is $name - date is `date` "
% echo ' my name is $name - date is `date` '
```

Help in linea

- Per fare riferimento alla documentazione on-line
  - `man` ***`command`***
  - `info` ***`command`***
  - `apropos` ***`keyword`***
- Per informazioni sull'utilizzo di `man` ed `info`:
  - `man man`
  - `info`

Leggere la documentazione

- **Nella documentazione e nei libri si trovano varie sintassi per la documentazione. Ad esempio:**

  `command` ***`[opt-arg] mandatory-arg {rep-arg}*`***

  - ***`opt-arg`*** è opzionale
  - ***`mandatory-arg`*** è obbligatorio
  - ***`rep-arg`*** può essere ripetuto **`n`** volte, con **`n`** >= 0, oppure si utilizza … per indicare argomento ripetibile
  - le opzioni (con **–**) possono raggruppate, e spesso non sono racchiuse fra parentesi quadre
  - corsivo per argomenti da sostituire

- **Esempi:**

  `uniq -c -`*`number`* **`[`** ***`inputfile`*** **`[`** ***`outputfile`*** **`] ]`**

  `mv [OPTION]... SOURCE... DIRECTORY`

Esercizi

- Qual é l'effetto del comando **`sort file > file`**, dove **`file`** è il nome di un file? E quello del comando **`sort file >> file ?`**
- Fare alcuni esperimenti per scoprire qual é l'effetto del comando **`tr str1 str2`** se le stringhe **`str1`** e **`str2`** hanno lunghezze diverse. Scrivere un comando per sostituire tutti i caratteri alfanumerici nell'input con un carattere **`<Tab>`**, in modo che non compaiano più **`<Tab>`** consecutivi.
- Il comando **`date`** fornisce data e ora su standard output. Studiarne la sintassi per estrarre soltanto l'ora.
- Estrarre da **`/etc/passwd`** il primo campo della prima riga

Esercizi

- Mediante **`ls`**, listare SOLO le subdirectory contenute in una directory assegnata.
- Trovare il modo per rimuovere un file conoscendo SOLO il suo *i-node*.
- Selezionare un editor e impadronirsi dei comandi base.
- Scrivere un file testo ASCII contenente una lista di comandi in sequenza per contare il numero di linee dei file il cui nome finisce per '**`.txt`**' presenti in una directory, il cui nome deve essere prelevato dalla variabile d'ambiente **`MYDIR`**.
  L'output deve andare sia su file che a video e deve contenere:
  - il pathname della working directory
  - il pathname della directory selezionata
  - il nome dei file '**`.txt`**' trovati e il loro numero
  - il numero di linee totale di tutti i file '**`.txt`**' trovati
  - il numero di tutti i file regolari nella directory
  - il numero di tutte le subdirectory nella directory
- Studiare i fondamenti di **`sed`** sulle slide e risolvere il quesito:
  mediante sed sostituire nel file di input tutte le cifre da 0 a 9 con -