Modulo 2: La shell

Laboratorio di Sistemi Operativi I
Anno Accademico 2006-2007

Francesco Pedullà
(Tecnologie Informatiche)

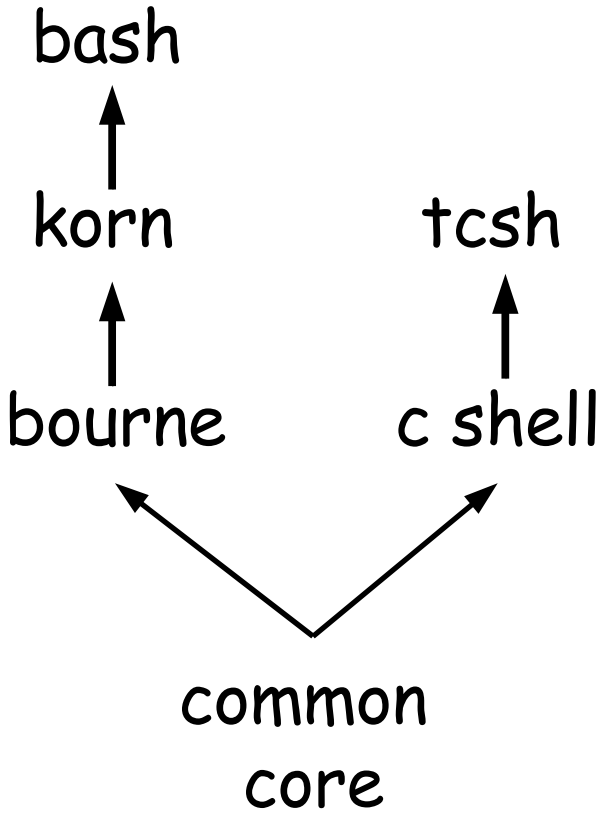
Massimo Verola
(Informatica)

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

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`

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”

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)

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:

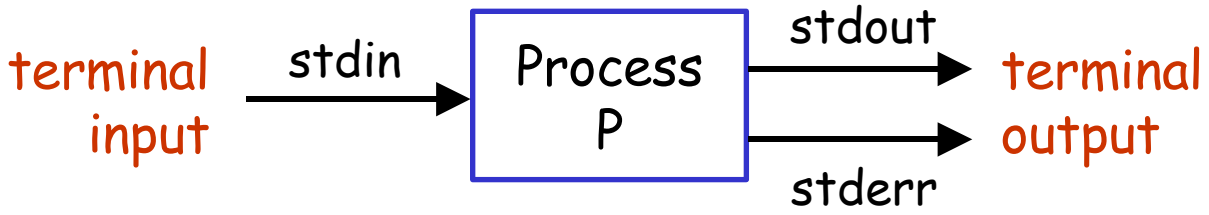
```
% firstname="Penguin"
% lastname="Blackwhite"
% echo $firstname $lastname
% export firstname
% bash
% echo $firstname $lastname
% firstname="Zebra"
% exit
% echo $firstname $lastname
```

- 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)
- **Redirezione dell'I/O e pipe permettono:**
  - "Disconnettere" questi stream dalle loro sorgenti/destinazioni abituali
  - "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`

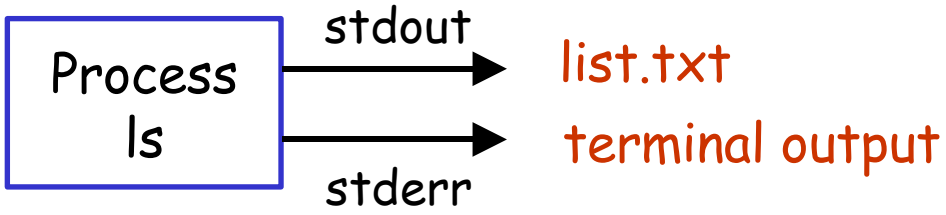
    ```
    mail penguin@antarctic.org < list.txt
    ```
  - Redireziona `stdout` e `stderr` del comando `rm` al file `/dev/null`

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

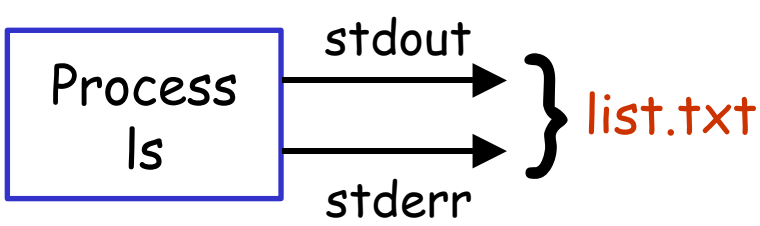
Redirezione dell'input/output (II)

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


```
ls > list.txt
```




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



Redirezione dell'input/output (III)

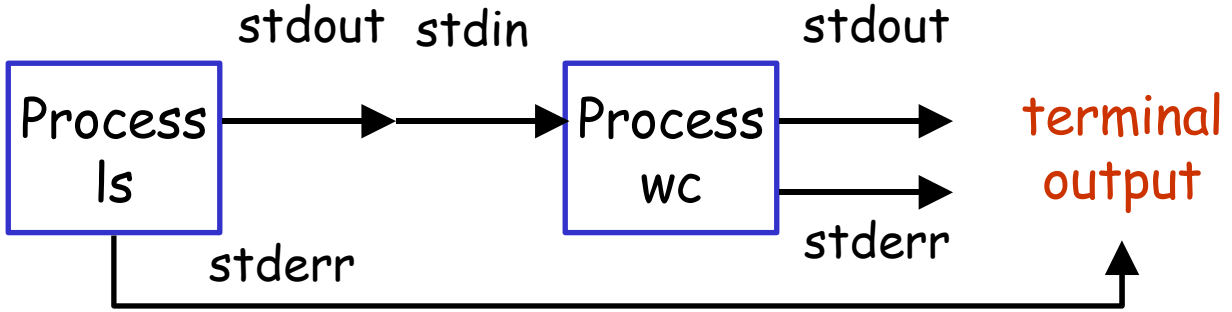
Altri esempi:

```
[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`**

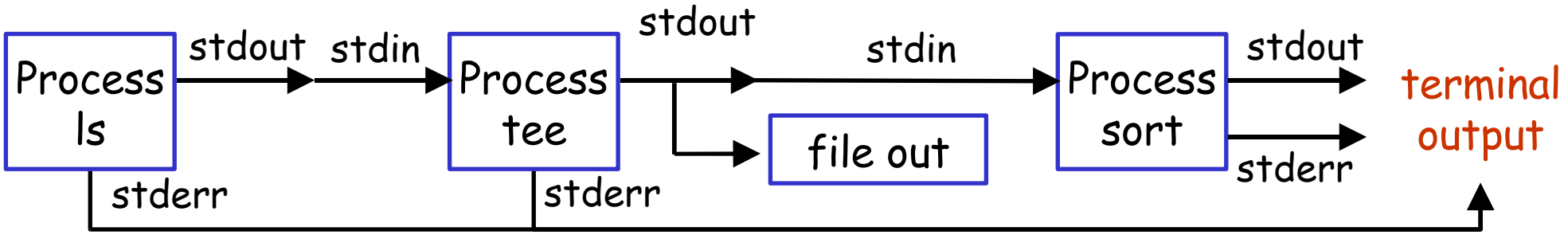
`sort -u` ordina in nomi ed elimina righe multiple con stesso nome

`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

- **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
    - `?` 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`** 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`** (suggerimento: comporre grep con **`find`** o **`xargs`**).

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 richiedonoi 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 (suggerimento: usare il comando **`head`** per estrarre la prima riga).

Appendice:

Breve rassegna di comandi (esterni) in ambiente Linux

Comandi: `cat` - `tac` - `rev`

- **Comando `cat` (concatenate):**
  - Stampa i file su stdout. Utilizzato con redirection, serve a concatenare file:

```
cat file.1 file.2 file.3 > file.123
```

- **Comando `tac` (inverso di cat):**
  - Stampa le linee dei file in ordine inverso, partendo dall'ultima linea
- **Comando `rev` (reverse):**
  - Stampa (invertite) le linee dei file, partendo dalla prima linea

Comando: `find` (I)

- **Comando: `find` *`pathlist expression`***
  - L'utility **`find`** analizza ricorsivamente i path contenuti in ***`pathlist`*** e applica ***`expression`*** ad ogni file
- **Sintassi di expression:**
  - ***`-name pattern`***

    True se il nome del file fa matching con pattern, che può includere i metacaratteri di shell: * [ ] ?
  - ***`-perm permission`***

    True se permission corrisponde ai permessi del file
  - ***`-print`***

    stampa il pathname del file e ritorna true

Comando: `find` (II)

- **Sintassi di expression:**
  - **`-ls`**

    Stampa gli attributi del file e ritorna true
  - **`-user` *`username`*`, -uid` *`userId`***

    True se il possessore del file è ***`username`*** / ***`userId`***
  - **`-group` *`groupname`*`, -gid` *`groupId`***

    True se il gruppo del file è ***`groupname`*** / ***`groupId`***
  - **`-atime | -mtime | -ctime -`*`count`***

    True se il file è stato “acceduto | modificato | modificato oppure cambiati gli attributi” negli ultimi ***`count`*** giorni

Comando: `find` (III)

- **Sintassi di expression:**
  - **`-type b | c | d | p | f | l | s`**

    True se il file è di tipo a blocchi | a caratteri | una directory | un named pipe | un file regolare | un link | un socket
  - **`-exec`** ***`command`***

    True se l'exit status di ***`command`*** è 0.

    `command` deve essere terminato da un "escaped ;" (\;).

    Se si specifica il simbolo {} come argomento di ***`command`*** , esso viene sostituito con il pathname del file corrente
  - **`-not, !, -a, -and, -o, -or`**

    Operatori logici (not, and, or)

Comando `find`: esempi

- `find . -name "*.c" -print`
  - Stampa il nome dei file sorgente C nella directory corrente e in tutte le sottodirectory
- `find / -mtime -14 -ls`
  - Elenca tutti i file che sono stati modificati negli ultimi 14 giorni
- `find . -name "*.bak" -ls -exec rm {} \;`
  - Cancella tutti i file che hanno estensione .bak

Comando: `xargs`

- **Comando: `xargs` *`command`***
  - **`xargs`** legge una lista di argomenti dallo standard input, delimitati da blank oppure da newline
  - esegue ***`command`*** passandogli la suddetta lista di argomenti
- **Esempio: concatena tutti i file `*.c`**

```
% find . -name "*.c" -print
./a.c
./b.c
% find . -name "*.c" -print | xargs cat > prova
```

Esempi: `find` e `xargs`

- `vi $(find . -name "*.java" | xargs grep -l "Alfa")`
  - Utilizza vi per visualizzare tutti i file nella directory corrente e nelle sottodirectory che hanno estensione java e contengono la parola "Alfa".
- `find ~ \( -name "*~" -o "#*#" \) -print | \`
  `xargs --no-run-if-empty rm -vf`
  - Rimuove tutti i file di backup o temporanei di emacs dalla home directory (ricorsivamente)
- `find . -type d -not -perm ug=w | xargs chmod ug+w`
  - Aggiunge il diritto di scrittura a tutti le directory nella directory corrente e nel suo sottoalbero

Esercizi

- E' possibile eseguire la funzione del primo esempio del lucido precedente senza usare il comando **`xargs`**?
- E' possibile eseguire la funzione del secondo esempio del lucido precedente senza usare il comando **`find`**?
- E' possibile eseguire la funzione del terzo script del lucido precedente senza usare il comando **`find`?**

Spazi e caratteri speciali

- `find ~ \( -name "*~" -o "#*#" \) -print0 | \`
  `xargs --no-run-if-empty --null rm -vf`
- **Attenzione agli spazi e ai caratteri speciali:**
  - **`xargs`** separa i nomi dei file tramite spazi o new line
  - il file **`relazione 1.txt`** viene trattato come due file
  - l'opzione **`-print0`** di **`find`** stampa stringhe null-terminated
  - l'opzione **`--null`** di **`xargs`** prende stringhe null-terminated
  - questa versione gestisce correttamente qualunque tipo di carattere speciale (come ")

Esempio

```
#!/bin/bash

# Move (verbose) all files in current directory

# to directory specified on command line.

if [ -z "$1" ];  then

   echo "Usage: `basename $0` directory-to-copy-to"

   exit 65

fi

ls . | xargs -i -t mv {} $1

 # This is the exact equivalent of mv * $1

 # unless any of the filenames has "whitespace"
   characters.

exit 0
```

Comandi per la gestione del testo

- **Comando: `head [-n] file`**
  - Lista le prime *n* linee di un file (10 default)
- **Comando: `tail [-n] file`**
  - Lista le ultime *n* linee di un file (10 default)
- **Comando: `cut`**
  - Un tool per estrarre campi dai file. Nonostante esistano altri tool (più sofisticati), **`cut`** è utile per la sua semplicità.
  - Due opzioni particolarmente importanti:
    - **`-d delimiter`**: specifica il carattere di delimitazione (tab default)
    - **`-f fields`**: specifica quali campi stampare

Comandi per la gestione del testo

- **Comandi: `expand, unexpand`**
  - L'utility **expand** converte i tab in spazi. L'utility **unexpand** converte gli spazi in *tab.*
- **Comando: `uniq`**
  - Questa utility rimuove linee duplicate (consecutive) dallo standard input. Viene usato spesso nei pipe con sort.
- **Comando: `sort`**
  - Ordina lo standard input linea per linea. E' in grado di eseguire ordinamenti lessicografici sulle linee, o di gestire l'ordinamento dei vari campi.
  - Ad esempio: l'opzione **-g** ordina in modo numerico secondo il primo campo dell'input

Esempio

```
% du -s /home/*
10000 /home/penguin
500 /home/black
2345 /home/white
26758 /home/gray
% du -s /home/* | sort -gr
26758 /home/gray
10000 /home/penguin
2345 /home/white
500 /home/black
```

Esempio (cont.)

```
% du -s /home/* | sort -gr | head -2

26758 /home/white

10000 /home/penguin

% du -s /home/* | sort -gr | head -2 | cut -f2

/home/white

/home/penguin

% for homedir in $(du -s /home/* | sort -gr | \
  head -2 | cut -f2); do echo $(basename $homedir) ; \
  done

white

penguin
```

Comandi per la gestione del testo

- **Comando: `wc` (word count)**
  - Conta linee, parole, caratteri
  - `-l | -w | -c` conta solo le linee | le parole | i caratteri
  - Certi comandi includono le funzionalità di **`wc`** come opzioni:

    `... | grep foo | wc -l` è equivalente a

    `... | grep --count foo`

- **Comando: `tr`**
  - Utility per la conversione di caratteri, seguendo un insieme di regole definite dall'utente:
  - Esempio: **`tr A-Z a-z < filename`**
    - Stampa filename trasformando tutti i caratteri in minuscoli
  - Esempio: **`tr -d 0-9 < filename`**
    - Stampa filename eliminando tutte i caratteri numerici

Esempio: filenames-to-lowercase

```
#! /bin/bash
# Changes every filename in working directory
# to all lowercase.
for filename in * ; do
  fname=`basename $filename`
  n=`echo $fname | tr A-Z a-z`
    # Change name to lowercase.
  if [ "$fname" != "$n" ] ; then
    # Rename only files not lowercase.
    mv "$fname" "$n"
fi
done
exit 0
```

Esempio: dos2unix

```
!/bin/bash

# dos2unix.sh: DOS to UNIX text file converter.

E_WRONGARGS=65

if [ -z "$1" -a -z "$2"]; then

  echo "Usage: `basename $0` file-source file-dest"

  exit $E_WRONGARGS

fi

CR='\015'  # Lines in DOS text files end in a CR-LF.

tr -d $CR < $1 > $2 # Delete CR and write to $2

exit 0
```

Comandi per il confronto di file

- **Comando: `cmp file1 file2`**
  - Ritorna true (exit status 0) se due file sono uguali, ritorna false (exit status != 0) altrimenti
  - Stampa la prima linea con differenze
  - Con l'opzione `-s` non stampa nulla (utile per script)
- **Comando: `diff file1 file2`**
  - Ritorna true (exit status 0) se due file sono uguali, ritorna false (exit status != 0) altrimenti
  - Stampa un elenco di differenze tra i due file (linee aggiunte, linee modificate, linee cancellate)
- **Comando: `diff dir1 dir2`**
  - Confronta due directory e mostra le differenze (file presenti in una sola delle due)

Comandi per la gestione di file

- **Comando: `locate, slocate, updatedb`**
  - I comandi **`locate`** e **`slocate`** (secure version di **`locate`**) cercano file utilizzando un database apposito. Il database riflette il contenuto del file system, ma va aggiornato con **`updatedb`** (solo root)
- **Comando: `file` *`file`***
  - Identifica il tipo di file a partire dal suo *magic number* (quando disponibile). L'elenco dei magic number si trova in **`/usr/share/magic`** (o altre posizioni, consultate info file)
  - Esempio:

```
% file prova.sh
prova.sh: Bourne-Again shell script text executable
```

Comandi per la gestione dei file

- **Comando: `basename` *`file`***
  - rimuove l'informazione del path da un pathname
- **Comando: `dirname` *`file`***
  - rimuove l'informazione del nome di file da un pathname
- **Nota: sono funzioni di stringa, non agiscono su un file effettivo**
- **Esempi:**

```
% basename /home/penguin/index.html
index.html
% dirname /home/penguin/index.html
/home/penguin
echo "Usage: `basename $0` arg1 arg2 ... argn"
```

Archiviazione e compressione

- **Compressione:**
  - Comandi: **`compress`**, **`uncompress`**
  - Comandi: **`gzip`**, **`gunzip`**
  - Comandi: **`bzip2`**, **`bunzip2`**
  - Comprimono e decomprimono file. **`compress`** è ormai in disuso (originario dei primi sistemi Unix); **`bzip2`** è meno diffuso (almeno per ora), ma è in alcuni casi più efficiente di **`gzip`**
- **Archivazione: tar**
  - Archivazione: **`tar zcvf`** ***`archive-name {file}*`***
    - **`z`**=comprimi, **`c`**=crea, **`v`**=verbose, **`f`**=su file
  - Estrazione: **`tar zxvf`** ***`archive-name`***
    - **`z`**=espandi, **`x`**=estrai, **`v`**=verbose, **`f`**=da file

Comandi per la gestione del tempo

- **Comando: `touch` *`file`***
  - utility per modificare il tempo di accesso/modifica di un file, portandolo ad un tempo specificato (**`touch -d`**)
  - può essere utilizzata anche per creare un nuovo file
  - creare file vuoti può essere utile per memorizzare solo la data
- **Comando: `date`**
  - Stampa informazioni sulla data corrente, in vari formati
- **Comando: `time` *`command`***
  - esegue il comando *command* e stampa una serie di statistiche sul tempo impiegato
  - Esempio: **`time find / -name "*.bak" -print`**

Introduzione alle espressioni regolari

- **Un'espressione regolare:**
  - è una stringa di caratteri e metacaratteri
  - i metacaratteri sono caratteri speciali che vengono interpretati in modo "non letterale" dal meccanismo delle espressioni regolari
- **Le espressioni regolari (Regular Expression, RE) sono utilizzate per ricerche e manipolazioni di stringhe**
  - Utilizzate da molti comandi quali **`grep`**, **`awk`**,**`sed`**, etc.
- **Definizione: *match***
  - Diciamo che una RE fa *match* con una particolare stringa se è possibile generare la stringa a partire dalla RE
- **Nota: Unix/Linux wildcard e RE hanno metacaratteri (e significati) differenti; non vanno confuse**

Espressioni regolari (I)

- **Sintassi:**
  - L'asterisco * fa match con qualsiasi numero di ripetizioni del carattere che lo precede, incluso 0
    - Esempio: "11*33" fa match con 1133, 11133, 111133, etc
  - Il punto . fa match con qualsiasi carattere, a parte newline
    - Esempio: "13.3" fa match con 13a3, 1303, ma non con 13\n3
  - Il caret ^ fa match con l'inizio di una linea (ma ha anche significati addizionali)
    - Esempio: "`^Subject:.*`" fa match con una linea di subject di posta elettronica
  - Il dollaro $ fa match con la fine di una linea:
    - Esempio: "`^$`" fa match con una linea vuota

Espressioni regolari (II)

- **Sintassi:**
  - Le parentesi quadre [...] sono utilizzate per fare match di un sottoinsieme (o di un intervallo) di caratteri
    - "[xyz]" fa match con i caratteri x, y, z
    - "[c-n]" fa match con qualsiasi carattere fra c ed n (nell'ordinamento ASCII)
    - "[a-zA-Z0-9]" fa match con qualsiasi caratter alfanumerico
    - "[0-9]*" fa match con qualsiasi stringa decimale
    - "[^0-9]" fa match con qualsiasi carattere non numerico
    - "[Yy][Ee][Ss]" fa match con yes, Yes, YES, yEs, etc.
- **Il backslash \ è usato come escape per i metacaratteri ed il carattere viene interpretato letteralmente: ad es., "\$"**
- **I metacaratteri perdono il loro significato speciale dentro [ ]**

Espressioni regolari (III)

- **Le espressioni regolari hanno un concetto di "parola":**
  - una parola è un pattern contenente solo lettere, numeri e underscore _
- **E' possibile fare matching**
  - con l'inizio di una parola: `\<`
    - Esempio: `\<Fo` fa match con tutte le parole che iniziano con Fo
  - con la fine di una parola: `\>`
    - Esempio: **ox\>** fa match con tutte le parole che finiscano con ox
  - con parole complete:
    - Esempio: `\<Fox\>`

Esempio: come cercare la parola **The** o **the** ?
RE #1: [tT]he ---> errata: troverebbe anche **other**
RE #2: □the□ ---> (□ *è blank*) errata: non troverebbe **the** all'inizio o fine di linea
RE #3: \<[tT]he>\ ---> OK: utilizza la sintassi \<pattern\>

Espressioni regolari (IV)

- **RE Recall**
  - un modo per riferirsi ai match più recente
- **Sintassi:**
  - per "marcare" una porzione di espressione regolare che volete riutilizzare: racchiuderla in `\(  \)`
  - per ripetere una porzione "marcata", si può utilizzare \n, con n=1..9
- **Esempi:**
  - `'^\([a-z]\)\1'`
    - la sintassi `\([a-z]\)` indica un pattern [a-z] che, una volta fatto il match, puo' essere indicato con la sintassi \1
    - quindi la RE fa match con tutte le linee che iniziano con due lettere minuscole identiche
  - Ricerca di parole palindrome di 5 lettere (esempio: radar): `'\([a-z]\)\([a-z]\)[a-z]\2\1'`

Espressioni regolari estese

- Il segno ? fa match con 0 o 1 ripetizioni della espressione regolare precedente
  - Esempio: “cort?o” fa match con coro e corto
- Il segno + fa match con 1 o più ripetizioni della espressione regolare precedente, ma non 0
  - Esempio: “ [0-9]+” fa match numeri non vuoti
- I segni \{ \} indicano il numero di ripetizioni che dalla espressione regolare precedente
  - Esempio: “[0-9] \{ 5\}” fa il match con tutti i numeri a 5 cifre
- I segni () servono a raggruppare espressioni regolari
  - Esempio: “(re)*” fa match con “”, re, rere, rerere, etc.
- Il segno | indica un’alternativa (or)
  - Esempio: “(bio|psico)logia” fa match con biologia e psicologia

Sommario RE (I)

Operator	*Effect*
.	Matches any single character.
?	The preceding item is optional and will be matched, at most, once.
*	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
{N}	The preceding item is matched exactly N times.
{N,}	The preceding item is matched N or more times.
{N,M}	The preceding item is matched at least N times, but not more than M times.
–	Represents the range if it's not first or last in a list or the ending point of range in a list.
^	Matches the empty string at the beginning of a line; also represents the characters not in the range of a list.
$	Matches the empty string at the end of a line.
\b	Matches the empty string at the edge of a word.
\B	Matches the empty string provided it's not at the edge of a word.
\<	Match the empty string at the beginning of word.
\>	Match the empty string at the end of word.

Sommario RE (II)

Pattern	*Matches*
^A	An A at the beginning of a line
A$	An A at the end of a line
A^	An A^ anywhere on a line
$A	A $A anywhere on a line
^\^	A ^ at the beginning of a line
^^	Same as ^\^
\$$	A $ at the end of a line
$$	Same as \$$
^.$	A line with any single character

Note that the caret ^ is only an anchor if it is the first character in a RE, while the $ is only an anchor if it is the last character in a RE.

Sommario RE (III)

Pattern	*Matches*
[0-9]	Any digit
[^0-9]	Any character other than a digit
[-0-9]	Any digit or a -
[0-9-]	Any digit or a -
[^-0-9]	Any character except a digit or a -
[ ]0-9]	Any digit or a]
[0-9]]	Any digit followed by a]
[0-9m-z]	Any digit or any character between m and z
[ ]0-9-]	Any digit, a -, or a]

Note that right square bracket] and dash – do not have special meaning if they directly follow a [.

Sommario RE (IV)

RE	*Matches*
*	Any line with a *
*	Any line with a *
\\	Any line with a \
^*	Any line starting with a *
^A*	Any line (useless RE !)
^A*	Any line starting with an A *
^AA*	Any line starting with one A
^AA*B	Any line starting with one or more A's followed by a B
^A\{4,8\}B	Any line starting with four, five, six, seven, or eight A's followed by a B
^A\{4,\}B	Any line starting with four or more A's followed by a B
^A\{4\}B	Any line starting with an AAAAB
\{4,8\}	Any line with a {4,8}
A{4,8}	Any line with an A{4,8}

*Note that modifiers like * and \{1,5\} only act as modifiers if they follow a character set.*

Sommario RE (V)

Hints for designing and analyzing a RE:

1. Knowing what it is you want to match and how it might appear in the text
2. Writing a pattern to describe what you want to match
3. Testing the pattern to see what it matches

Hints for evaluating the results of a pattern-matching operation:

1. ***Hits***: the lines I wanted to match
2. ***Misses***: the lines I didn't want to match
3. ***Misses that should be hits***: the lines that I didn't match but wanted to match
4. ***Hits that should be misses***: the lines that I matched but I didn't want to match

Comando `grep`

- **Il comando `grep` permette di cercare pattern in un insieme di file**
  - se non vi sono file specificati, la ricerca è nello standard input
  - il pattern è espresso come espressione regolare
  - tutte le linee che contengono il pattern vengono stampate su stdout
  - in caso di file multipli, le linee sono precedute dal pathname del file che le contiene (a meno di opzione **`-h`**)
  - opzione **`-v`**
    - fa in modo che l'output contenga tutte le linee che non contengono il pattern
  - opzione **`-R`**
    - analizza ricorsivamente le subdirectory
  - opzione **`-c`**
    - conta le occorrenze, invece di stamparle
  - opzione **`-i`**
    - ignora distinzioni tra caratteri maiuscoli e minuscoli
  - opzione **`-q`**
    - modo "silente" (*quiet*), non scrive nulla su stdout

Comando `grep`

- **Esistono quattro versioni:**
  - `grep [options] pattern {file}*`
    - pattern è una espressione regolare
  - `fgrep [options] pattern [file(s)]`
    - pattern è una stringa fissa
    - versione più veloce
  - `egrep [options] pattern [file(s)]`
    - pattern è una espressione regolare estesa
  - `zgrep [options] pattern [file(s)]`
    - è anche in grado di cercare in file compressi (compressed or gzipped)

Comando grep - Esempi

```
if echo "$VAR" | grep -q txt ; then
   echo "$VAR contains the substring \"txt\""
fi
```

```
grep '[Ff]irst' *.txt
file1.txt:This is the first line of file1.txt.
file2.txt:This is the First line of file2.txt.
```

Esempi: cosa fanno queste ricerche?

- `grep -c 'ing$' /usr/dict/words`
- `grep -c '^un.*$' /usr/dict/words`
- `grep -ic '^[aeiou]' /usr/dict/words`
- `grep -ic '\(.\)\1\1' /usr/dict/words`
- `grep -c '^\(..\).*\1$' /usr/dict/words`
- `grep -ic '^\(.\)\(.\).\2\1$' /usr/dict/words`