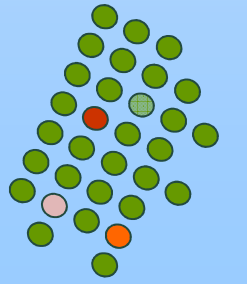
BASH:
Bourne Again Shell
(2)

Capitolo 1 -- Rosenblatt



Filename e wildcard

- Wildcard: permette di specificare più file
  - ? qualsiasi carattere (1 solo)
  - * qualsiasi sequenza di caratteri (0 o +)
  - [*set*] qualsiasi carattere in *set*
  - [!*set*] qualsiasi carattere non in *set*
- ?, *, !, [e] sono caratteri speciali

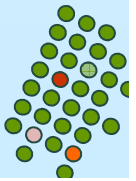
Filename e wildcard: esempi

*.txt	tutti i file che finiscono in .txt
p?ppo	pippo, poppo, pappo, p1ppo, ...
*.t[xyw]t	file che finiscono in .txt, .tyt, .twt
[!abc]*	file che non iniziano con a op b op c
*[0-9][0-9]	file che finiscono con 2 cifre
[!a-zA-Z]*	file che non iniziano con una lettera

I/O

- I/O sotto Unix è basato su 2 idee:
  - un file I/O è una sequenza di caratteri
  - tutto ciò che produce o accetta dati è trattato come un file (inclusi i dispositivi hardware)
- standard file
  1. standard input (stdin)
  2. standard output (sdtout)
  3. standard error (stderr)
- alcuni comandi:

cat	copia l'input nell'output
grep	ricerca una stringa nell'input
sort	ordina le linee dell'input
cut	estrae colonne dall'input



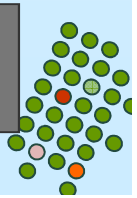
Ridirezione

- il comando cat usato senza argomenti, prende input da stdin e manda l'output a stdout

```
bash> cat
Questa è una linea di testo.
Questa è una linea di testo.
^D
```

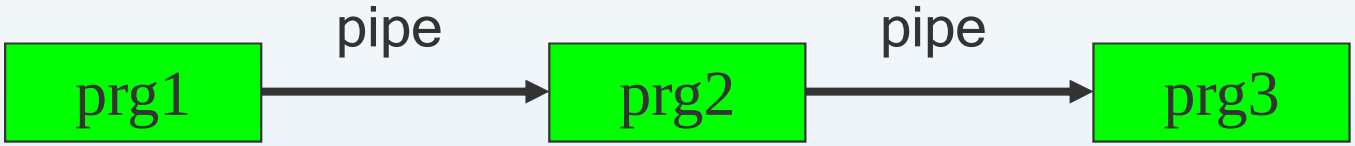
- < file1 fa si' che l'input sia preso da file1
- > file2 fa si' che l'output vada nel file2
- >> file come prima, ma append se file esiste

```
bash> cat < file1 > file2
```

Pipeline

- Pipeline: serve a mandare l'output di un programma nell'input di un altro programma



- La barra verticale **|** rappresenta una pipe

- cut: prende il primo campo (-f1) usando il carattere (**:**)
  come separatore (-d:)

```
bash> cut –d : -f 1 /etc/passwd | sort | lp
```

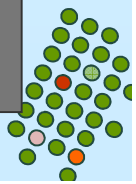
rescigno:LM1cU6tgSYecKb:601:100:Adele Rescigno:/home/rescigno...

Processi in background

- Un processo e' un programma in esecuzione
- Quando eseguiamo un programma dalla shell dobbiamo aspettare che il programma termini
- Se volessimo far eseguire un programma che non necessita input dall'utente ed intanto volessimo mandare in esecuzione altre cose => mandiamo il processo relativo al programma in background: &
- jobs: informazioni sui processi in background

```
bash> tar –xzf archivio.tgz &
[1] 3156
bash>
```

```
bash>
[1]+ Done              tar –xzf archivio.tgz
```

I/O in processi in background

- Processi in background non dovrebbero avere I/O
- Il terminale e’ uno solo
- Un solo processo puo’ usufruirne: foreground
- Se un processo in background fa I/O
  1. Se vuole input si blocca aspettando
  2. Ouput: viene mescolato su video con quello che stiamo facendo
- Si possono usare le ridirezioni per evitare il problema

Caratteri speciali

1. ~ home directory
2. # commento
3. $ precede il nome di variabile
4. & processo in background
5. ? * [ ] per wildcard (nomi file)
6. | pipe
7. ( ) inizio e fine subshell

Caratteri speciali

8.	{ }	blocco di comandi
9.	;	separatore di comandi
10.	’	quote (virgoletta) - forte
11.	"	quote (virgolette) - debole
12.	< >	ridirezione
13.	!	simbolo di negazione
14.	/	(slash) separatore directory nel nome del file
15.	\	(backslash) simbolo di “escape”

Virgolette (quoting)

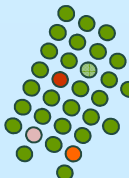
- A volte si vogliono usare i caratteri speciali senza il loro significato speciale

```
bash> echo 2 * 3 > 5 espressione vera
bash> _
```

Otteniamo

- un file nome: "5"
- contenuto: "2", seguito dai nomi dei file nella cwd (*) e da "3 espressione vera"

```
bash> echo '2 * 3 > 5 espressione vera'
2 * 3 > 5 espressione vera
bash> echo "2 * 3 > 5 espressione vera"
2 * 3 > 5 espressione vera
```

Backslash escaping

- Il backslash toglie il significato speciale al carattere che lo segue

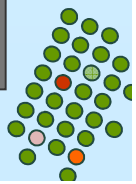
```
bash> echo 2 \* 3 \> 5 espressione vera
2 * 3 > 5 espressione vera
```

- Per il backslash **** : **’ \’** oppure ****
- Per il doppio quote **"** : **\"** all’interno di **" ... "**

```
bash> echo "\"2 * 3 \> 5\" espressione vera"
"2 * 3 > 5" espressione vera
```

- Per il quote **’** : \’ all’interno di ’ ... ’ puo’ dare problemi, ma ...lo si può rafforzare **’\’’**

```
bash> echo ’L’\’’aquila non volava.’
L’aquila non volava.
```

Caratteri di controllo

- tasto CTRL + un’altro tasto (normalmente non stampano niente sullo schermo ma il sistema operativo li interpreta come comandi speciali)

1. CTRL-C interrompe il processo (SIGINT)
2. CTRL-\ interrompe il processo (SIGQUIT)
3. CTRL-Z sospende il processo (SIGSTP)
4. CTRL-D fine input (EOF)
5. CTRL-S sospende output video
6. CTRL-Q ripristina output video

- Sono di uso comune, ma non obbligatorio
  - stty –a
  - fornisce informazioni sui caratteri di controllo

Bash: Introduzione

- help: manuale online

```
bash> help cd
cd: cd [-PL] [dir]
    Change the current directory to DIR.  The variable $HOME is the
    default DIR.  The variable CDPATH defines the search path for
    the directory containing DIR.  Alternative directory names in CDPATH
    are separated by a colon (:).  A null directory name is the same as
    the current directory, i.e. `.'.  If DIR begins with a slash (/),
    then CDPATH is not used.  If the directory is not found, and the
    shell option `cdable_vars' is set, then try the word as a variable
    name.  If that variable has a value, then cd to the value of that
    variable.  The -P option says to use the physical directory structure
    instead of following symbolic links; the -L option forces symbolic
    links to be followed.
bash>
```

- help re, equivalente a help 're*'
  - tutti i comandi che iniziano per re

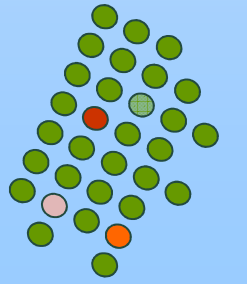
echo e read

- echo e read sono le printf e scanf della shell
- echo "messaggio",
  - stampa il messaggio
- read var1 var2
  - legge valori di var1 e var2

```
bash> echo –n "Digita i valori di due variabili:"; read a b
Digita i valori di due variabili: pippo pluto
bash> echo $a
pippo
bash> echo $b
pluto
bash>
```

Editor di linea

Capitolo 2 -- Rosenblatt



Editor di linea

- Potete usare vi o emacs
  - set -o emacs
  - set -o vi

emacs:	vi:	
CTRL-A	0	inizio linea
CTRL-E	$	fine linea
CTRL-F	l (elle)	un carattere a destra
CTRL-B	h	un carattere a sinistra
CTRL-D	x	cancella un carattere
ESC-B	b	spostati di una parola indietro
ESC-F	w	spostati di una parola in avanti
ESC-DEL	db	cancella una parola indietro

emacs

`$ fgrep -l Duchess < ~cam/book/alice_`	CTRL-A
`$ fgrep -l Duchess < ~cam/book/alice`	CTRL-D
`$ grep -l Duchess < ~cam/book/alice`	ESC-F
`$ grep -l Duchess < ~cam/book/alice`	ESC-F
`$ grep -l Duchess < ~cam/book/alice`	ESC-D
`$ grep -l _ < ~cam/book/alice`	Scrivi Cheshire
`$ grep -l Cheshire_< ~cam/book/alice`	Dai ENTER

Nota che per emacs una parola è una stringa di caratteri alfanumerici

Editor di linea

- Normalmente l'editing mode è settato per la bash con quello di emacs
- Quando si setta l'editing mode a vi
  - se si scrive normalmente sulla linea di comando è come se si usasse **i** (insertion mode)
  - se invece si ci vuole spostare sulla linea su cui si è digitato si deve prima dare ESC e poi i comandi su citati
  - per poter poi inserire di nuovo testo dare
    - i
    - a se si vuole inserire dopo il carattere corrente
    - A se si vuole inserire alla fine della linea

vi

`$ fgrep -l Duchess < ~cam/book/alice_`	ESC, 0
`$ fgrep -l Duchess < ~cam/book/alice`	x
`$ grep -l Duchess < ~cam/book/alice`	$
`$ grep -l Duchess < ~cam/book/alice_`	b
`$ grep -l Duchess < ~cam/book/alice`	B
`$ grep -l Duchess < ~cam/book/alice`	I
`$ grep -l Duchess < ~cam/book/alice`	R, scrivi resciano
`$ grep -l Duchess < ~rescigno/book/alice`	ESC, ENTER

Nota che se si digita B e W ci si riferisce ad una word non-blanc

Editor di linea

- History
  - file: .bash_history ($HISTFILE)

emacs:	vi:	
CTRL-P	k o -	comando precedente
CTRL-N	j o +	prossimo comando
CTRL-R	/string	cerca all'indietro
CTRL-S	?string	cerca in avanti
ESC-<	1 G	primo comando
ESC->	n G	ultimo comando

History: emacs

- Si supponga che dopo ore di lavoro si voglia richiamare un comando

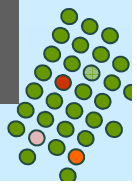
  per esempio: `fgrep –l Duchess < ~cam/book/alice`

  e che non si voglia scorrere l'intero history file; si può ricorrere alla ricerca dando

  - CTRL-R
  - Quello che era scritto sulla linea scompare ed appare

    `(reverse-i-search)' '`
  - Scriviamo ciò che vogliamo cercare, cioè fgrep, ed otterremo

```
$ (reverse-i-search) 'fgrep': fgrep –l Duchess < ~cam/book/alice
```

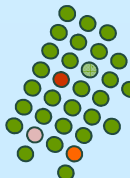
History: vi

- Si supponga che dopo ore di lavoro si voglia richiamare un comando (per esempio: `grep –l Duchess < ~cam/book/alice`) e che non si voglia scorrere l'intero history file; si può ricorrere alla ricerca dando
  - ESC (per entrare in control mode)
  - Scriviamo /^grep (per ricercare all'indietro le sole linee che cominciano con grep) ed otteniamo

```
$ grep –l Duchess < ~rescigno/book/alice
```

  - Che non è quanto volevamo; allora diamo n fino a quando non compare

```
$ grep –l Duchess < ~cam/book/alice
```

Bash: history

history visualizza tutti i comandi

- Selezionatore di comandi
  - ! inizia una ricerca nella “history”
  - !! esegue il comando precedente
  - !47 esegue il 47-esimo comando
  - !-23 comando corrente – 23
  - !*str* ultimo comando che inizia per *str*
  - !?*str*? ultimo comando che contiene *str*
  - ^*s1*^*s2* ultimo comando, sostituisce *s1* con *s2*

Bash: history

- Selezionatore di Token (parole)
  - n (n+1)-esima parola (0 = prima)
  - ^ primo argomento (seconda parola)
  - $ ultimo argomento
  - x-y tutte le parole tra l'x-simo e lo y-simo argomento

Bash: history

- Il selezionatore di parole deve seguire il selezionatore di comando dopo un **:**
  - !!:0 prima parola del comando precedente
  - !!:$ ultima parola del comando precedente
  - !!:3-6 dalla 4ª alla 7ª parola
  - !!:* tutte le parole tranne la prima
  - !!:2-* dalla 3ª all'ultima parola