Modulo di Sistemi Operativi per il corso di Master
RISS: Ricerca e Innovazione nelle Scienze della Salute
Unisa, 17-26 Luglio 2012



Sistemi Operativi

Introduzione all'uso e alla programmazione di una shell

Bash: Bourne Again Shell

Shell: interfaccia testuale (CLI)

E' un programma che interpreta comandi

Viene eseguito al login

Termina al logout

Bash e' standard con Linux

Altre shell: sh, tcsh, ksh

Bash: Bourne Again Shell

Sviluppata, a partire dal 1988, nel progetto GNU

Progetto GNU: produrre una versione gratuita di UNIX GNU = Gnu's Not Unix

Steve Bourne ha scritto una delle prime shell per UNIX (1979): sh

Bash: Bourne Again Shell, in tributo a S. Bourne

Bash: Bourne Again Shell

bash> sort –n num_telefoni > num_ordinati

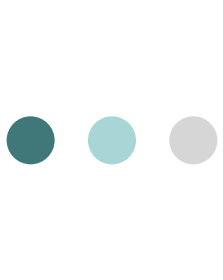
Compito della shell

1. Separare gli elementi (token) del comando:
   - sort, -n, num_telefoni, >, num_ordinati

2. Determinare il significato dei token
   - sort comando da eseguire
   - -n e num_telefoni argomenti
   - > operatore di redirezione
   - num_ordinati nome del file per la redirezione

3. Preparare il sistema a redirigere l'output

4. Cercare ed eseguire il comando



Bash: Bourne Again Shell

bash> sort –n num_telefoni > num_ordinati

Parole separate da spazi o TAB

La prima parola è il nome del comando

Seguono gli argomenti

Un' opzione è un argomento speciale che modifica il comportamento prestabilito (default) del comando

Directory e primi comandi

current working directory (cwd)

- al login, cwd = home directory
- per conoscere la cwd: **pwd**
- per cambiare la cwd: **cd**

Carattere tilde:

~: home directory

~user: home directory di "user"

Esempi

```
bash> cd ~pippo
bash> pwd
bash> cd ~
```

Directory e primi comandi

I file di una directory

- . è un riferimento alla directory stessa
- .. è un riferimento alla directory padre
- **ls** elenca i file presenti

Alcune opzioni:

-a: all (tutti gli elementi, anche quelli "nascosti")

-l: long (con informazioni varie)

Esempi

```
bash> ls -a
bash> ls -l
bash> ls -la
```

Filename e wildcard

Wildcard: permette di specificare più file

- ? qualsiasi carattere (uno solo)
- * qualsiasi sequenza di caratteri (zero o più di uno)
- [set] qualsiasi carattere in set
- [!set] qualsiasi carattere non in set

?, *, !, [e] sono caratteri speciali

Filename e wildcard

*.txt tutti i file che finiscono in .txt

p?ppo tutti i file con nomi pippo, poppo, pappo, p1ppo, …

*.t[xyw]t tutti i file che finiscono in .txt, .tyt, .twt

[abc]* file che iniziano con a o b oppure c

[!abc]* file che non iniziano con a o b oppure c

*[0-9][0-9] file che finiscono con 2 cifre

[!a-zA-Z]* file che non iniziano con una lettera

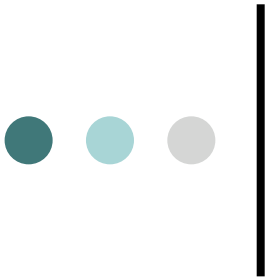
I/O: standard file e redirezione

Standard I/O file

1. standard input (stdin) tastiera
2. standard output (sdtout) schermo
3. standard error (stderr) schermo

Comando cat

1. senza argomenti, legge da stdin ed invia su stdout
2. < file_in, legge l'input da file_in
3. > file_out, invia l'output su file_out
4. >> file_out, aggiunge alla fine di file_out



Comandi echo e read

bash> **echo** stampa questa frase a video

bash> **echo** stampa il contenuto della variabile $SHELL

bash> **read** var

bash> **echo** $var

echo permette di visualizzare testo o il contenuto di una variabile
read permette di memorizzare dati in una variabile

Pipeline

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



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

  robdep:LM1cU6tgSYecKb:601:100:Roberto De Prisco:/home/robdep...

- La barra verticale | rappresenta una pipe

```
bash> cut -d: -f1 /etc/passwd | sort | lp
```

Processi in background

- Quando eseguiamo un programma dalla shell dobbiamo aspettare che il programma termini
- Processi in background: &

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

- informazioni sui processi in background

```
 bash>
 [1]+ Done                tar -xzvf archivio.tgz
```

I/O processi in background

- Processi in background non dovrebbero avere I/O
- Il terminale è uno solo
- Un solo processo può usufruirne: foreground
- Se un processo in background fa I/O
  1. Se vuole input si blocca aspettando
  2. Output: su video, mischiato con altre attività che stiamo facendo
- Si possono usare le ridirezioni per evitare il problema

Caratteri speciali

- Alcuni caratteri hanno significati 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
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

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

- Per il backslash stesso: ' \' oppure \\

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

- \' all'interno di ' ... ' può dare problemi, ma ...

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

Caratteri di controllo

- tasto CTRL + un altro tasto

1. CTRL-C interrompe il processo (SIGINT)
2. CTRL-\ interrompe il processo (SIGQUIT)
3. CTRL-Z sospende il processo (SIGTSTP)
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: help

```
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>
```

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

- 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 (controllare)
ESC-<	1 G	primo comando
ESC->	n G	ultimo comando

Bash: history

- history — visualizza tutti i 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**

- Token (parole)
  - n — (n+1)-esima parola (0 = prima)
  - ^ — primo argomento (seconda parola)
  - $ — ultimo 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

- Modificatori
  - !cat:2-*/p stampa senza eseguire
  - !!/s/old/new sostituisce old con new

Variabili di ambiente

- Controllano il modo in cui la shell si comporta
- Forniscono informazioni ai processi

```
HOME= usr/local/home/pd
USER=pd
HOSTNAME=macbook-di-paolo-darco-2
SHELL=/bin/bash
PATH=/bin:/usr/bin:/usr/local/bin/:.
...
```

- Alcune sono assegnate automaticamente
- Possiamo definire nuove variabili
- Possiamo cambiare il valore delle variabili

File speciali

- .bash_profile, .bashrc, .bash_logout
- .bash_profile:
  - eseguito ad ogni login

```
PATH=bin:/usr/bin:/usr/local/bin/:.
EDITOR=/usr/bin/vi
PS1='\h:\w> '
PS2='> '
export EDITOR
.....
```

  - Qualsiasi modifica ha effetto al prossimo login
  - source .bash_profile
  - . .bash_profile

File speciali

- .bashrc:
  - eseguito per ogni subshell

- .bash_profile deve contenere comandi che devono essere eseguiti solo al login
- .bashrc deve contenere i comandi che vogliamo in ogni subshell

- .bash_logout
  - eseguito al logout

Alias

- alias nome=comando
- alias cerca=grep
  - cerca roberto numeri_telefonici.txt
- alias lf='ls -F'
- alias printall='cat *.txt | lpr'
- alias mali=mail
- alias emcas=emacs
- Non si possono usare parametri. Esistono le funzioni per questo

Opzioni

- Cambiano il comportamento della shell
- set -o **nome** opzione diventa ON
- set +o **nome** opzione diventa OFF

- Alcuni esempi:
  - set -o emacs editor di linea: emacs
  - set -o ignoreeof CTRL-D non termina la shell
  - set -o noclobber non sovrascrive file
    - sort file1 > file2
  - set -o noblog non espande i nomi dei file
  - set -o nounset da' errore se variabile non definita

- shopt permette di controllare altre opzioni

Variabili

- Definire una variabile: **nome=valore**
  - non ci devono essere spazi intorno al segno =
  - se **valore** contiene spazi bisogna usare "**valore**"

- Usare una variabile: ${**nome** }
  - le {} possono essere omesse se non ci sono ambiguita'
  - OK se il nome della variabile non e' seguito da
    - lettera, numero o underscore
    - $UID_pippo non va bene se vogliamo riferirci a UID
    - ${UID}_pippo

- Cancellare una variabile: unset **nome**
  - una variabile che non e' definita vale "" (stringa nulla)
  - con l'opzione nounset, l'uso di una variabile non definita causa un errore

Variabili

```
bash> PATH=/bin:/usr/bin:/usr/local/bin/:.
bash> echo "il path e' $PATH"
il path e' /bin:/usr/bin:/usr/local/bin/:.
bash> echo 'il path e\' $PATH'
il path e' $PATH
bash>
bash> nome=pippo
bash> echo ${nome}.txt
pippo.txt
bash> echo $altronome
bash> set -o nounset
bash> echo $altronome
bash: altronome: unbound variable
bash>
```

Variabili di ambiente

Sono variabili predefinite (che possiamo cambiare)

- HOME — home directory
- PATH — ricerca dei comandi
- EDITOR — vi o emacs (normalmente)
- PS1 — prompt
- HISTFILE — nome del file per l'history
- PWD — current working directory (path)
- OLDPWD — directory precedente
- SECONDS — contati dall'inizio della shell
- IFS — separatori (internal field separator)
- ...

Variabile per il prompt

- PS1="[\u@\h]> "
  - [paodar@macbook-di-paolo-darco-2]>

- Personalizzare il prompt
  - \d data
  - \h,\H hostname
  - \s nome della shell
  - \t,\T orario
  - \u nome utente
  - \v,\V versione di bash
  - \w,\W cwd
  - \!,\# il numero del comando (history)
  - \\ backslash
  - \nnn carattere in ottale

Usare più linee

- Comando = una linea
- Per usare più linee basta usare \ per indicare che si vuole continuare sulla linea successiva
- Se ci sono delle virgolette aperte è automatico

```
bash> echo comando che  \
>usa due linee.
comando che usa due linee.
bash> echo "comando che
>usa due linee."
comando che
usa due linee.
bash>
```

- PS2 è il prompt per le linee successiva alla prima

Variabili

- I processi che hanno come padre la shell possono accedere alle variabili di ambiente
  - HOME
  - PATH
  - PWD ...

- Le altre variabili devono essere "esportate" altrimenti non sono visibili

- export **nome**
- export **nome =valore**

- **nome =valore comando**
  - TERM=vt100 emacs file1

Script

```
#!/bin/bash
echo Questo e\' uno script
echo I file in /bin sono:
ls /bin
```

- Uno script è un file con comandi shell
- Per eseguirlo
  - source **filename**
  - Oppure
    - chmod +x **filename**
    - **./filename**
  - Il secondo metodo lancia una subshell ed esegue lo script nella subshell
- Prima riga #!**comando**, opzionale
  - viene usato **comando** per interpretare lo script

Funzioni

- Lo stesso concetto dei linguaggi di programmazione

```
function nome {
        comandi shell
}

oppure

nome () {
        comandi shell
}
```

- Possono essere definite anche fuori dagli script

- declare –f dà la lista delle funzioni definite

Parametri

- I parametri sono passati in variabili speciali
- Valgono sia per lo script che per le funzioni
- $1, $2, ..., ${10}, ${11}, ...
  - $0 e' il nome dello script

Esempio

```
#!/bin/bash
function stringa {
 echo il parametro passato alla funzione stringa e\' $1
}
echo il nome dello script e\' $0
stringa Ottimo
```