LABORATORIO DI SISTEMI OPERATIVI

BASH: standard stream, redirezioni, altri comandi di uso comune, variabili

fabio.rossi@unipg.it

075 5855005

BASH: command-line completion

- *bash* ha la funzione di completamento delle linee di comando premendo il tasto *TAB*
- mentre si scrive, alla prima pressione di *TAB*:
  1. se c'è una sola possibilità il comando viene completato
  2. altrimenti il comando viene eventualmente completato fin dove è possibile farlo in modo univoco e viene emesso un segnale sonoro
- nel caso 2., premendo una seconda volta, viene mostrata la lista delle scelte possibili

BASH: command-line completion

```
bash:~$ cat /etc/hos Tab
            cat /etc/host  e segnale sonoro Tab
            cat /etc/host  e segnale sonoro Tab
host.conf    hostname     hosts        hosts.allow
hosts.deny
bash:~$ cat /etc/host
```

BASH: gestione della history

- *BASH* tiene nota degli ultimi *n* comandi dati alla shell
- *n* dipende dalle variabili *HISTFILESIZE* e *HISTSIZE* (*manbash*)

history	visualizza gli ultimi *n* comandi eseguiti (il numero di comandi visualizzato e memorizzato dipende dalle variabili HISTSIZE e HISTFILESIZE)
!a	esegue l'ultimo comando nella history che inizia per 'a'
!68	esegue il comando numero 68 della history
!!	ripete l'ultimo comando eseguito

P.S.: sono comandi interni

BASH: metacaratteri

- "A character that, when unquoted, separates words. A *metacharacter* is a space, tab, newline, or one of the following characters: ‘|’, ‘&’, ‘;’, ‘(’, ‘)’, ‘<’, or ‘>’." (da *Bash Reference Manual*)
- per la shell hanno un significato, li "interpreta"
- vi sono altri caratteri che la shell interpreta:

  '*', '?', Ctrl+d (*EOF*), Ctrl+c, '`', etc.
- ne parleremo via via…

BASH: standard streams

- ogni processo ha dei canali predefiniti di "comunicazione" con il mondo esterno
- ***stdin, stdout*** e ***sdterr.*** Di default questi tre stream sono collegati al terminale di controllo (se ce n'è uno...) del processo che sta eseguendo il comando
- lo *standard output* è utilizzato per mostrare i risultati della computazione (tipicamente a video)
- lo *standard error* è utilizzato per comunicare informazioni addizionali sullo stato della computazione, ad esempio degli errori (tipicamente a video)
- lo *standard input* è utilizzato come sorgente da cui prendere i dati da elaborare (tipicamente da tastiera)

BASH: standard streams

- ***stdin***, ***stdout*** e ***sdterr*** possono essere "collegati" ad altro con la ***redirezione*** e il ***pipeline***, per esempio a dei file
- ogni file aperto nel sistema è identificato da un descrittore di file (*fd*) che è un intero positivo
- ***stdin*** ha come *fd* 0, ***stdout*** 1 e ***sdterr*** 2

- la Bash permette di ridirigere qualsiasi descrittore

BASH: redirezione dell'input

comando [n]< filename

- il descrittore *n* è associato a *filename* che viene aperto in lettura.
- Se *n* è omesso lo *standard input* è associato al file, cioè il comando legge dal file l'input

BASH: redirezione dell'input

Esempi:

il file *testo.txt* contiene le seguenti righe:

```
fabio
ciccio
pippo
```

```
bash:~$ sort < testo.txt
ciccio
fabio
pippo
```

BASH: redirezione dell'output

comando [n]> filename

- il descrittore *n* è associato a *filename* che viene aperto in scrittura
- Se *n* è omesso lo *standard output* è associato al file, cioè il comando scrive sul file l'output

N.B.: se *filename* esiste il file viene sovrascritto... (a rigore ciò dipende dall'opzione *noclobber*...)

BASH: redirezione dell'output

Esempi:

- **`bash:~$ ls > lista_file.out`**

  lista_file.out conterrà l'elenco degli oggetti contenuto nella directory corrente

- **`bash:~$ cat > lista_file.out`**

  `a` *inseriti dall'utente*

  `b` ..

  `c` ..

  `^d` ..

  `bash:~$`

  lista_file.out conterrà le righe:

  `a`

  `b`

  `c`

BASH: redirezione di *stderr*

riprendiamo:

comando [n]> filename

- basta porre n=2

comando 2> filename

BASH: redirezione dell'output in append

comando [n]>> filename

- il descrittore *n* è associato a *filename* che viene aperto in scrittura. Se *filename* non esiste viene creato altrimenti i dati vengono aggiunti alla fine del file

- Se *n* è omesso lo *standard output* è associato al file

BASH: redirezioni: esempio

- la directory corrente contiene i seguenti file: *mesi*, *pippo*, *pluto*, *paperino*
- il file *mesi* contiene le seguenti righe:

  gennaio
  febbraio
  aprile

```
bash:~$ ls cip
ls: cannot access cip': No such file or directory
bash:~$ cat mesi > log
bash:~$ ls cip 2>> log
bash:~$ cat log
gennaio
febbraio
aprile
ls: cannot access 'cip': No such file or directory
bash:~$ sort < log > log_ordinato
bash:~$ cat log_ordinato
aprile
febbraio
gennaio
ls: cannot access 'cip': No such file or directory
bash:~$
```

BASH: redirezione simultanea di *stdout* e *stderr*

- Due sintassi disponibili:

  command &> filename

  oppure

  command >& filename

  la prima è preferibile. Semanticamente equivale a:

  command > filename 2>&1

- analogamente si realizza *l'append*:

  command &>> filename

  Semanticamente equivale a:

  command >> filename 2>&1

BASH: Pipeline

cmd1 | cmd2 | … | cmdN

- vari comandi (anche solo due) separati dal carattere '|' (*pipe*)
- *stdout* di *cmd1* viene connesso allo *stdin* di *cmd2* e così via di seguito
- al posto del carattere '|' si può utilizzare '|&' con cui *stdout* e *sdterr* del comando precedente vengono entrambi connessi all'input del comando successivo
- ogni comando è eseguito in un processo differente (*subshell*)
- esempio: **`ps ax | grep init`**

BASH: comandi informativi

id	visualizza gli ID di utente e gruppo
hostname	visualizza/imposta l'*hostname* della macchina
uname	visualizza informazioni sul sistema
which	visualizza il path dell'eseguibile che esegue un comando
date	visualizza/imposta data e ora correnti
who, w	visualizzano chi è loggato nel sistema
tty	visualizza il nome del terminale
df	visualizza l'occupazione dello spazio disco

BASH: trattamento di testi

grep (-rvec)	visualizza linee che soddisfano criteri (*patterns)* di ricerca
sort (-nf)	Ordina le linee di un file di testo
awk	implementa un linguaggio di programmazione per il «*pattern scanning and processing*»
tr	sostituisce, comprime e/o cancella caratteri
uniq	visualizza o sostituisce lineee ripetute
cut	seleziona parti di linee di un file (es.: a campi delimitati)
sed (-ei)	*stream editor* per filtrare e modificare un testo
wc (-l)	conta linee, parole e caratteri

BASH: trattamento di testi

more	visualizza il contenuto di un file una pagina alla volta
less	permette di navigare in un file una pagina alla volta (ed anche altro...)
diff	confronta il contenuto di due file per linee

BASH: gestione processi

ps (axuj)	visualizza i processi attivi
top	visualizza in modo interattivo processi e risorse occupate
nice	esegue un comando modificandone la priorità
renice	modifica la priorità di un processo
kill	invia un *signal* ad un processo
strace (-p)	monitora *system calls* e *signal* di un processo

BASH: altri comandi

find	cerca file che corrispondono a determinati criteri
date	visualizza e modifica l'ora corrente
uptime	risponde a: "*da quanto la macchina è accesa?*"
last	riporta le ultime login
lastlog	riporta le ultime login di tutti o di un singolo utente
xargs	da *man*: "*build and execute command lines from standard input*"
su (-, -c)	permette di "agire" come altro utente
echo (-en)	visualizza una linea di testo (interno)
help	fornisce la lista dei comandi built-in della shell (interno)
tee	"scrive" il suo *stdin* su *stdout* e uno o più file

BASH: altri comandi

basename	visualizza in output il solo nome file senza le precedenti directory da un pathname. Se richiesto rimuove il suffisso. Es.: *basename /home/user1* visualizza "*user1*"
dirname	visualizza le directory che precedono il nome del file in un pathname Es.: *dirname /home/user1* visualizza "*/home*"
sleep n	attende n secondi senza fare nulla (che detta così...)

BASH: Backtick '`' (ASCII 96)

- Backtick o apice inverso, accento grave, apice inclinato, ecc.
- in BASH le stringhe contenute tra caratteri backtick sono sostituite dall'output del comando che rappresentano:

```
user1@host1:~$ pwd
/home/user1
user1@host1:~$ ls -ld `pwd`
drwxr-xr-x 93 user1 group1 12288 mar 17 11:52 /home/user1
user1@host1:~$
```

BASH: Esecuzione in background 1

- Esegue un comando senza interazione con l'utente
- La shell viene subito «liberata», cioè non devo attendere la fine del comando per impartirne altri
- *&* alla fine del comando
- Il processo in *background* continuerà comunque a mandare l'output al terminale
- I processi in *background* sono detti «*jobs*»
- Eseguendo un comando in background la *shell* mostra il *PID* del processo ed il codice del job

BASH: Esecuzione in background 2

- Comando *jobs* per visualizzare i job in esecuzione
- *fg [%x]* per far tornare nella modalità interattiva (*foreground*) un job. *%x* è facoltativo ed *x* è il codice del job (1,2,… etc.)
- *Kill -9 <pid>* per terminare un job, dove *<pid>* è il *PID* del processo
- Anche *ps* aiuta…
- Per mandare un processo da *foreground* in *background*:
  - Stopparlo con *Ctrl+z*
  - Mandarlo in *background* col comando *bg*

BASH:Variabili

- array di coppie nella forma *nome=valore.*
  - *nome*: stringa alfanumerica che comincia per lettera
  - *valore*: stringa di caratteri

- si dichiarano/assegnano:
  - **`VAR1=2`** # senza spazi intorno all'uguale
    # per convenzione il nome in maiuscolo
  - **`VAR1=\`pwd\``** # '`' è l'apice inverso (ASCII 96)
  - **`VAR1=$VAR2`**

BASH:Variabili

- si referenziano con **$NOME**

- si può utilizzare anche la notazione **${NOME}**. Utile se può esserci ambiguità. Es.:

```
VAR1="NUMBER "
VAR1ONE="scratch"
echo $VAR1ONE              # cosa viene visualizzato?!
```

BASH:Variabili

- Es.:

```
user1@host1:~$ VAR1="NUMBER "
user1@host1 :~$ VAR1ONE="scratch"
user1@host1 :~$ echo $VAR1ONE
scratch
user1@host1 :~$ echo ${VAR1}ONE
NUMBER ONE
user1@host1 :~$
```

BASH:Variabili

- nelle costanti stringa delimitate da DOPPI apici **$NOME** viene interpretato
- nelle costanti stringa delimitate da SINGOLI apici **$NOME** non viene interpretato
- Es.:

```
VAR1="33"
echo "$VAR1"          # visualizza '33'
echo '$VAR1'          # visualizza '$VAR1'
```

BASH: ambiente (environment)

- variabili impostate inizialmente dalla *BASH* quando viene invocata in base ai file di configurazione

- "ereditate" da ogni comando eseguito dalla *BASH*

BASH: ambiente (environment)

- **`NOME=VALORE`** per definire una nuova variabile **`$NOME`**
- **`export NOME`** per renderla disponibile ai processi *child*
- **`unset NOME`** per cancellarla
- **`set`** o **`env`** per visualizzare l'environment. **`set`** mostra tutte le variabili e funzioni definite nella shell
- **`set`** permette anche di settare gli attributi (opzioni) che regolano il funzionamento della BASH

BASH: alcune variabili d'ambiente predefinite

- *PATH*: contiene una lista di directory, separate da ':', in cui la BASH cerca i comandi da eseguire (se non viene specificato un path per essi)
- *HOSTNAME*: nome host del sistema
- USER: nome dell'utente
- PWD: directory corrente