ESD Electronic Systems Design

Synthesis Verification Testing Power Communication

Introduzione alla Bash di Linux

Nicola Drago
Nicola Dall'Ora

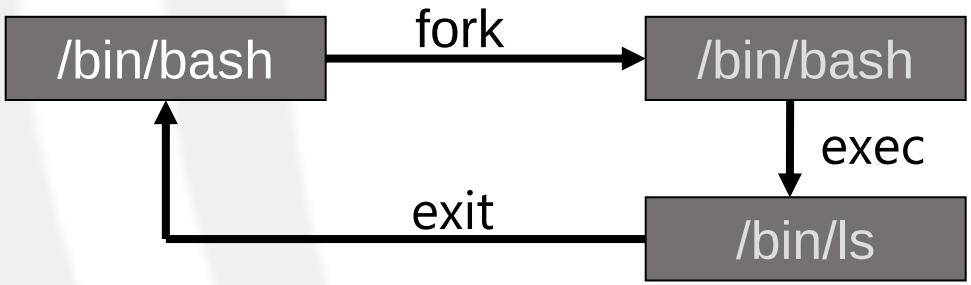
Anno 2023-23

Dipartimento di Informatica

Università di Verona, Italy

Funzionamento della bash

- Esempio: esecuzione del comando ls



- System call coinvolte:
  - fork
    - Processo padre Bash crea un nuovo processo figlio
  - exec
    - Processo figlio carica il codice del comando richiesto
  - exit
    - Processo figlio termina
  - wait
    - Processo padre Bash aspetta la terminazione del figlio

Sintassi comandi

- Sintassi generale di un comando LINUX

  comando [-opzioni] [lista argomeniti]

- Si possono dare più comandi sulla stessa riga separandoli con ';' (saranno eseguiti in sequenza)
  $comando_1$; $comand_2$; ...; $comando_N$

- Ogni comando Linux ha una manual page
  man <comando>

File system

File system

- Visualizzare il contenuto di una directory

  ls [-opzioni] <path2Dir>

  Opzioni

  -a visualizza anche i file nascosti

  -l output in formato esteso

  -r ordine alfabetico inverso

  -F appende carattere per indicare il tipo di file
  (/ for directories, * for executables, @ for links)

  -R elenca anche i file nelle sottodirectory

Path

- . è la directory corrente
- .. è la directory padre di quella corrente
- Path assoluto = /dir1/dir2/…
  - Parte della radice ‘/’ del file system
- Path relativo = dir1/dir2/…
  - Parte della cartella corrente

File

Tipi di file in Linux

- Directory
  - Contiene file e altre sotto directory
- Special file
  - Entry point per un dispositivo di I/O
- Link
  - Collegamento ad un file
- File ordinario
  - Tutti gli altri file

Visualizzazione di file

- cat file
  - Stampa il contenuto del file fornito su standard output
- head [-n] file
  - visualizza le prime 10 righe

    -n k visualizza le prime *k* righe
- tail [-n] file
  - visualizza le ultime 10 righe

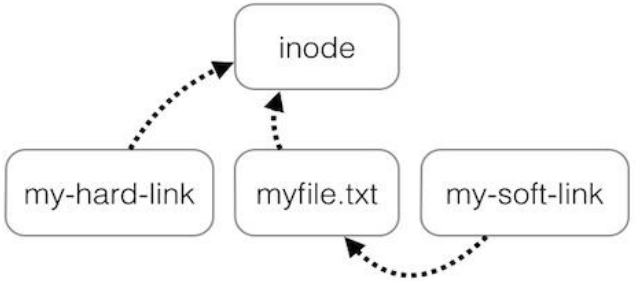
    -n k visualizza le ultime k righe

Manipolazione di file

- cp file dest
  - Copia file in dest
- mv file dest
  - Sposta/rinomina file in dest
- rm [-fr] file/direcotry ...
  - Cancella il file/directory fornita

    f force, **r** recursive (esempio parametri)

Link



- Hard link
  - Un nome che punta a un i-node
- Soft link
  - Un nome che punta a un Hard-link
- Nota
  - Un file viene rimosso quando tutti i suoi hard link sono stati rimossi

Directory

`cd <path>`

cambia la directory in quella indicata

`pwd`

mostra il path assoluto della directory corrente

`mkdir dirName`

crea una nuova directory

`rmdir dir`

cancella la directory fornita (deve essere vuota)

Cambio di proprietario/gruppo

`chgrp [-R] gruppo file`

- cambia il gruppo del file

`chown [-R] utente[:gruppo] file`

- cambia proprietario [e gruppo] del file

`chmod [-R] <permessi> file/directory`

- cambia i permessi di un file/directory

L'opzione `-R` indica di propagare il comando anche alle sottodirectory

Cambio protezione

r w x
(421)

- chmod 640 prova.txt
  - Lettura/scrittura per proprietario (6 = (4)r + 2(w)) [chmod u+rw prova.txt]
  - Lettura per gruppo (4 = 4(r)) [chmod g+r prova.txt]
  - Nessun permesso per altri (0)
- chmod 755 dir
  - Lettura/scrittura/esecuzione per proprietario (7=4(r) + 2(w) + 1(x))
  - Lettura/esecuzione per gruppo (5=4(r)+1(x))
  - Lettura/esecuzione per altri (5=4(r)+1(x))

Ricerca di un file

find directory espressione

Visita tutto l'albero sotto la directory specificata e ritorna i file che rendono vera l'espressione

- name pattern (usare gli apici se si usano espressioni regolari)
- type tipo (b c d l f)
- user utente
- group gruppo
- newer file
- atime, mtime, ctime [+/-] giorni
- print
- size [+/-] blocchi

Confronto di file

diff [-opzioni] file1 file2
diff [-opzioni] dir1 dir2

mostra le righe diverse, indicando quelle da aggiornare (a), cancellare (d) e cambiare (c)

-b ignora gli spazi a fine riga, collassa gli altri

-i ignora la differenza tra maiuscolo e minuscolo

-w ignora completamente la spaziatura

Confronto di file – Esempio

- Prova1
  ciao
  come va?
  Bene
  grazie

- Prova 2
  ciao
  come?
  bene
  molto bene
  grazie

- Prova 3
  ciao

```
$ diff Prova1 Prova2
2c2
< come va?
---
> come?
4c4,5
< grazie
---
> molto bene
> grazie
```

```
$ diff Prova1 Prova3
2,4d1
< come va?
< bene
< grazie
```

```
$ diff Prova3 Prova1
1a2,4
> come va?
> bene
> grazie
```

Processi

Lo stato dei processi

- Il comando `ps` fornisce uno snapshot dei processi correnti di sistema

- Output esempio

```
PID    TTY          TIME         CMD
3490   pts/3        00:00:00     bash
3497   pts/3        00:00:00     ps
```

`PID`	Process Identifier
`TTY`	terminale da cui il processo è eseguito
`TIME`	tempo totale di esecuzione
`CMD`	comando eseguito corrispondente

Lo stato dei processi

- Opzioni principali
  - a: visualizza tutti i processi
  - x: tutti i processi dell'utente
  - r: tutti i processi nello stato running

- Stati di un processo:
  - **R** running/in running queue
  - **T** stopped (es. ^Z)
  - **S** interruptible (awaiting a event to complete)
  - **D** uninterruptible (usually I/O)
  - **Z** zombie

Gestione dei processi

- I processi normalmente eseguono in **foreground** e hanno tre canali standard connessi al terminale: stdin, stdout, e stderr
- I processi eseguiti con **&** eseguono in **background** e sono privi di stdin (es. ./a.out &)
- Un processo in foreground può essere sospeso con **^Z**

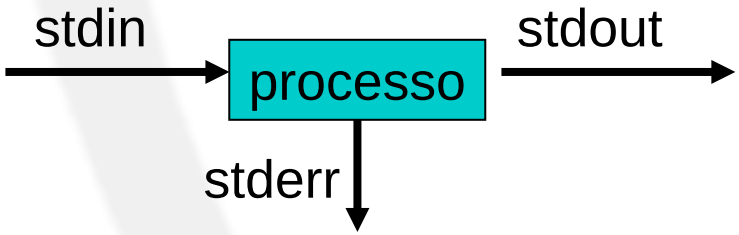
Gestione dei processi – Comandi

- `jobs [-l]`
  elenca i processi in background o sospesi. Esempio output:
  [1]+ Stopped sleep 2
- `bg [%job]`
  riprende i processi specificati in background
- `fg [%job]`
  riprende i processi indicati in foreground
- `kill [-signal] PID`
  manda un segnale al processo indicato
  (i più comuni SIGKILL, SIGTERM)

La programmazione della Bash

Redirezione dell'I/O

- Ogni processo ha tre canali associati



- Ogni canale può essere rediretto
  - su file
  - su un altro canale

Redirezione dell'I/O

comando < file

stdin letto da file

comando > file

stdout scritto file

comando >> file

stdout aggiunto in coda al file

comando 1> file_out 2> file_error

stdout su file_out e stderr su file_error

comando1 | comando2

pipe tra comando1 e comando2.
stdout del comando1 viene ridiretto come stdin per il comando2

Variabili d'ambiente

- La Bash ha un insieme di variabili d'ambiente. Ogni variabile rispetta il pattern: variabile=valore

Le principali variabili d'ambiente

- `PWD  : path corrente sul filesystem`
- `SHELL: path alla Bash`
- `USER : username dell'utente`
- `HOME : path della home directory dell'utente`

Variabili d'ambiente

Come leggere la variabili dell'ambiente:

- printenv [variabile]
  - stampa il valore della variabile d'ambiente fornita
- env
  - stampa tutte le variabili d'ambiente
- echo $variabile
  - stampa il valore della variabile d'ambiente fornita

Bash script

- Uno script è una lista di comandi di sistema eseguiti sequenzialmente dalla Bash

- Esecuzione
  - Eseguendo sulla linea di comando:

    ```
    bash script [argomenti]
    ```

  - Eseguendo direttamente uno `script`
    - E' necessario che lo script file abbia il permesso di esecuzione
    - La prima riga dello script file inizia con **`#!`**, seguita dal path delle Bash (`#!/bin/bash`)

Bash script

```
#!/bin/bash
# i caratteri dopo ‘#’ sono commenti

date     #restituisce la data
who      #restituisce chi è connesso
```

myScript.sh

Variabili speciali

- La bash memorizza gli argomenti della linea di comando dentro una serie di variabili
  `$1, ... $n`
- Alcune variabili speciali

`$$`	PID del processo shell
`$0`	Il programma corrispondente al processo corrente
`$#`	il numero di argomenti
`$?`	se esistono argomenti (no=0, si=1)
`$*`, `$@`	tutti gli argomenti

Bash – Variabili

- Per acquisire un valore da standard input

```
read variabile
```

- Esempio

```
read x
< pippo
echo $x
> pippo
```

Bash - Variabili

- NOTA
  - I valori delle variabili sono sempre STRINGHE
  - Per valutazioni aritmetiche si deve usare l'operatore `$(())`

```
x=0
echo $x+1
> 0+1
echo $((x+1))
> 1
```

Bash – Uso output di un comando

- E' possibile utilizzare l'output di un comando come valore di inizializzazione per una variabile
- Sintassi
  - `variabile=`comando``
  - `variabile=$(comando)`
- Esempio:
  - tutti i file contenuti nella directory corrente:

```
lista_file=`ls`    # lista_file=$(ls)
```

Bash – Uso output di un comando

- E' possibile utilizzare l'output di un comando come "dati" all'interno di un altro comando
- Tramite l'operatore "` `"
- Sintassi
  - `` `comando` `` (` = ALT+96 su tastiera italiana)
  - `$(comando)`
- Esempio
  - Cancellazione di tutti i file con il nome **test.log** contenuti nell'albero delle directory **/home/joe**

```
rm `find /home/joe -name test.log`
```

Bash – Strutture di controllo

- Strutture condizionali

```
if [ condizione ];
  then comandi;
fi
```

```
if [ condizione ];
  then comandi;
elif [ condizione ];
  then comandi;
…
else
  comandi;
fi
```

Bash – Test e condizioni

- Operatori per una condizione (`man test` per altri)

Operatore	Vero se …	#operandi
-n	operando ha lunghezza ≠ 0	1
-z	operando ha lunghezza = 0	1
-d	esiste una directory con nome = operando	1
-f	esiste un file regolare con nome = operando	1
-e	esiste un file con nome = operando	1
-r, -w, -x	esiste un file leggibile/scrivibile/eseguibile	1
-eq, -ne	gli operandi sono interi e sono uguali/diversi	2
=, !=	gli operandi sono stringhe e sono uguali/diversi	2
-lt, -gt	operando1 <, > operando2	2
-le, -ge	operando1 ≤, ≥ operando2	2

Bash – Strutture di controllo

- Esempio:

```
PATH2FILE= “$HOME/.bash_profile”

if [ -e $PATH2FILE ]; then
    echo “you have a .bash_profile file”;
else
    echo “you have no .bash_profile file”;
fi
```

Bash – Strutture di controllo

- Ciclo for

```
for arg in lista; do
    comandi
done
```

- `lista` può essere
  - un elenco di valori
  - una variabile (corrispondente ad una lista di valori separati dal carattere spazio)

Bash – Strutture di controllo

- Esempi

```
lista=`ls`
for file in $lista
do
   ls -l $file
done
```

****

```
LIMIT=10
 # Valutazioni aritmetiche richiede (())
for ((a=1; a <= LIMIT; a++))
do
   echo $a
done
```

Bash – Strutture di controllo

- Ciclo while

```
while [ condizione ]
do
  comandi
done
```

Bash – Strutture di controllo

- Esempio

```
LIMIT=10
a=1
while ((a <= LIMIT))
do
    echo $a
    a=$((a+1))
done
```

Bash – Filtri

- Programmi che ricevono dati di ingresso da stdin e generano risultati su stdout
- Molto utili assieme alla ri-direzione dell'I/O
- Alcuni dei filtri più usati sono

```
more
sort
grep, fgrep, egrep
cut
head, tail
uniq
wc
awk (sed)
```

Bash – grep

- Per cercare se una stringa compare all'interno di un file

```
grep [-opzioni] pattern file
```

Opzioni

`-c`	conta le righe che contengono il pattern
`-i`	ignora la differenza maiuscolo/minuscolo
`-l`	elenca solo i nomi dei file contenenti il pattern
`-n`	indica il numero d'ordine delle righe
`-v`	considera solo righe che non contengono il pattern

Bash – Espressioni regolari

- I pattern di ricerca in grep possono essere normali stringhe di caratteri o espressioni regolari. In questo caso, alcuni caratteri hanno un significato speciale (a meno che siano preceduti da \)

`.`	un carattere qualunque
`^`	inizio riga
`$`	fine riga
`*`	ripetizione (zero o più volte)
`+`	ripetizione (una o più volte)
`[ ]`	un carattere tra quelli in parentesi
`[^]`	un carattere esclusi quelli in parentesi
`\<`	inizio parola
`\>`	fine parola

Bash – Varianti di grep

`fgrep [option] [string] [file] …`

- I pattern di ricerca sono stringhe
- E' veloce e compatto

`egrep [option] [string] [file] …`

- I pattern di ricerca sono delle espressioni regolari estese
- E' potente ma lento
- Richiede molta memoria

Bash – Ordinamento di dati

`sort [-opzioni] file …`

Opzioni

`-b`	ignora gli spazi iniziali
`-d`	(modo alfabetico) confronta solo lettere, cifre e spazi
`-f`	ignora la differenza maiuscolo/minuscolo
`-n`	(modo numerico) confronta le stringhe di cifre in modo numerico
`-o file`	scrive i dati da ordinare in **`file`**
`-r`	ordinamento inverso
`-t carattere`	usa **`carattere`** come separatore per i campi
`-k S1,S2`	usa i campi dalla posizione S1 alla S2

Bash – Selezione di Campi

```
cut -clista file
cut -flista [-dchar] [-s] file
```

- **`lista`** specifica un intervallo del tipo
  - **`a,b`** significa '**`a`**' e '**`b`**'
  - **`a-b`** significa da '**`a`**' a '**`b`**'

Bash – Selezione di Campi

- Opzioni
  - `-c` seleziona per caratteri
  - `-f` seleziona per campi
    Il campo è definito dal separatore
    (default carattere TAB)
  - `-dchar` **`char`** è usato come separatore
  - `-s` considera solo le linee che contengono il separatore
- Esempi

```
cut -c1-12 file
```
prende i primi 12 caratteri di ogni riga del file

```
cut -c1, 4 file
```
prende il campo 1 e 4 di ogni riga del file

```
cut -f1-4 file
```
prende i primi 4 campi di ogni riga del file

Bash – Selezione di Campi

- Altri esempi

```
cut -d: -f1,5 /etc/passwd
```

Estrae user e nome completo degli utenti

```
ps -x | cut -d" " -f1
```

Elenca i PID dei processi nel sistema

Bash – wc

`wc [-c] [-l] [-w] file`

Legge i file nell'ordine e conta il numero di caratteri, linee e parole

Opzioni

`-n`	conta solo i caratteri
`-l`	conta solo le righe
`-w`	conta solo le parole

- Esempio

  `ps -x | tail +2 | wc -l`

  Conta il numero di processi attivi (tail +2 per togliere l'intestazione)

Bash – uniq

uniq [-u][-c] file

- Trasferisce l'input sull'output sopprimendo duplicazioni contigue di righe
- Assume che l'input sia ordinato
- Opzioni
  - **-u** visualizza solo righe non ripetute
  - **-c** visualizza anche il contatore del numero di righe