Corso di Laurea in Informatica A.A. 2023-2024

Laboratorio di Sistemi Operativi

Alessandra Rossi

Grep e le espressioni regolari

grep [*opzioni*] *pattern* [*nomefile*]

- Stampa le righe del file che corrispondono al pattern
- Il pattern è una *espressione regolare*
- Nel caso più semplice, il pattern può essere una stringa senza caratteri speciali:

```
grep a pippo.txt
```

stampa le righe di pippo.txt che contengono una a

grep [*opzioni*] *pattern* [*nomefile*]

- Se nomefile non è specificato, legge da standard input
- Questo consente la concatenazione in *pipe*

`ls -l | grep 2006`	elenca i file che sono stati modificati l'ultima volta nel 2006 (ma non solo ...)
`ls -l | grep rwx`	elenca i file per cui almeno una categoria di utenti ha tutti i permessi (ma non solo ...)

- Con opzione “-v”, stampa le righe che non corrispondono al pattern

`ls -l | grep -v doc`	elenca i file che non contengono “doc” nel nome

- Con “-c”, visualizza solo il numero di occorrenze della stringa nel file; “-i” case-Insensitive; “-n” numero di riga

grep [*opzioni*] *pattern* [*nomefile*]

```
lso:~>grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin

lso:~>grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
12:operator:x:11:0:operator:/root:/sbin/nologin

lso:~>grep -c root /etc/passwd
2
```

grep [*opzioni*] *pattern* [*nomefile*]

```
lso:~>grep -v bash /etc/passwd |grep -v nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
news:x:9:13:news:/etc/news:

lso:~>grep -c account /etc/passwd
0
lso:~>grep -c -i account /etc/passwd
1
lso:~>grep -i account /etc/passwd
lso:x:501:501:LSO Account:/home/lso:/bin/bash
lso:~>grep -i account /etc/passwd |wc -l
   1
```

Basic Regular Expressions

- Una espressione regolare e' un *pattern che descrive un insieme di stringhe*
- *L'elemento atomico delle espressioni regolari e' il carattere*
  - *Un carattere e' una espressione regolare che descrive se stesso*
  - *L'espressione “a” descrive “l'insieme di stringhe {a}”*
- *La maggior parte dei caratteri sono “espressioni regolari”*
- *I Metacaratteri corrispondono ad operatori*
  - *Un metacarattere puo' essere utilizzato con il suo valore utilizzando il carattere di escape “\”*

Basic Regular Expressions

. (un punto)	qualunque carattere	(1)
exp*	zero o più occorrenze di exp	(2)
^exp	exp all'inizio del rigo	(1)
exp**$**	exp alla fine del rigo	(1)
[a-z**]**	un carattere nell'intervallo specificato	
[^a-z**]**	un carattere fuori dall'intervallo	

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Basic Regular Expressions

\<exp	exp all'inizio di una parola	(1)
exp**\>**	exp alla fine di una parola	(1)
exp{N}	exp compare N volte	(1)
exp{N,}	exp compare almeno N volte	(1)
exp{N,M}	exp almeno N volte e al più M	(1)
[[:CLASS:]]	un carattere in CLASS	(1)

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Basic Regular Expressions

Le classi di caratteri POSIX:

[[:alpha:]]	I caratteri alfabetici
[[:alnum:]]	I caratteri alfanumerici
[[:digit:]]	Le cifre
[[:upper:]]	I caratteri alfabetici maiuscoli
[[:lower:]]	I caratteri alfabetici minuscoli

Esempi

a*b	zero o più “a” seguite da una “b”
a.*b	una “a” prima di una “b”
\<[[:upper:]]	una parola che inizia con lettera maiuscola
^d	la lettera “d” all'inizio del rigo
^a*$	un rigo vuoto o composto solo di “a”
^a.*b$	un rigo che inizia con “a” e finisce con “b”
\<.-	una parola con un trattino al secondo posto

Basic Regular Expressions

Molti comandi per l'elaborazione di testi di UNIX (ad esempio grep, ed, sed, ..) consentono la definizione di espressioni regolari, ossia di schemi per la ricerca di testo basati sull'impiego di metacaratteri:

- Generalmente, i metacaratteri usati da tali comandi non coincidono con i metacaratteri impiegati dalla shell per identificare i nomi dei file
- Molti caratteri che hanno un significato speciale nelle espressioni regolari hanno pure un significato speciale per la shell

- Attenzione a non confondere i metacaratteri di shell con quelli che non lo sono
- Utilizzare gli apici ' ' o i doppi apici " " per racchiudere le espressioni

Basic Regular Expressions

- La “concatenazione” di espressioni regolari e' una espressione regolare:
  - Le “stringhe” possono essere costruite dalla “concatenazione” dei caratteri.
  - Una stringa corrisponde (“match”) ad una concatenazione di stringhe se e' composta da due sottostringhe che corrispondono, rispettivamente, alle due espressioni regolari
  - “ab” corrisponde alla concatenazione di exp1=“a” ed exp2=“b”
- L'operatore “|” (es. exp3=exp1|exp2)
  - Una stringa corrisponde ad exp3 se esiste un match con exp1 o con exp2.

Extended Regular Expressions

exp**+**	una o più occorrenze di exp	(1)
exp**?**	zero o una occorrenza di exp	(2)
exp1 **|** exp2	exp1 oppure exp2	(2)
(exp **)**	equivalente a exp, serve a stabilire l'ordine di valutazione	

In **grep**, questi simboli vanno preceduti da "\" (backslash)
In **egrep** (*extended grep*), si usano direttamente

Note: (1) è un carattere normale per Bash
(2) ha un significato diverso per Bash

Esempi per grep

```
[[:digit:]]\
+
```

una sequenza non vuota di cifre

```
^a\|b
```

un rigo che inizia con a oppure contiene b (precedenza)

```
^\(a\|
b\)
```

un rigo che inizia con a oppure con b

```
\(\(\.txt\)\|\(\.doc\)\)
\>
```

una parola che termina con .txt o con .doc

(in egrep, “`((\.txt)|(\.doc))\>`”)

Esempi per grep

```
lso:~>egrep '^r.*n$|^r.*37' /etc/passwd
rpm:x:37:37::/var/lib/rpm:/bin/bash
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

```
lso:~>grep '^r.*n$|^r.*37' /etc/passwd
lso:~>grep '^r.*n$\|^r.*37' /etc/passwd
rpm:x:37:37::/var/lib/rpm:/bin/bash
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
```

Esercizi

1. Elencare i file con permesso di esecuzione per il proprietario
2. Elencare le directory il cui nome inizia per maiuscola
3. Elencare i file con permesso di esecuzione oppure di scrittura per il gruppo di appartenenza

Risultati

1. Elencare i file con permesso di esecuzione per il proprietario

   ls -l | grep "^...x"

   ls -l | grep "^...x" | grep -v "^d" - senza directory

2. Elencare le directory il cui nome inizia per maiuscola

   ls -d */ | grep ^[[:upper:]]

   ls -l | grep ^d | grep -o "\w*$" | grep ^D - ma qui non prende i nomi con caratteri speciali

3. Elencare i file con permesso di esecuzione oppure di scrittura per il gruppo di appartenenza

   ls -l | grep “^……x\|^.....w"

   ls -l | grep “^……x\|^.....w" | grep -v “^d”

Riferimenti

Capitolo 4 di [Bash Guide for Beginners]

I processi BASH

I processi

- I processi sono programmi in esecuzione
- lo stesso programma può corrispondere a diversi processi (es., tanti utenti che usano emacs)
- Ogni processo può generare nuovi processi (figli)
- Ogni processo ha
  - process identification number (**PID**)
  - parent process identification number (**PPID**)
- Tranne il processo **init**, che ha PID=1 e nessun PPID

(La radice della gerarchia di processi è il processo init con PID=1. init è il primo processo che parte al boot di sistema).

Processi

Un programma singolo, nel momento in cui viene eseguito, è un ***processo***.

La nascita di un processo, cioè l'avvio di un programma, può avvenire solo tramite una richiesta da parte di un altro processo già esistente.

Si forma quindi una sorta di gerarchia dei processi organizzata ad albero.

Il processo principale (*root*) che genera tutti gli altri, quello dell'eseguibile **init** che a sua volta attivato direttamente dal kernel.

Tabella Processi

Il kernel gestisce una tabella dei processi che serve a tenere traccia del loro stato. In particolare sono registrati i valori seguenti:

- il nome dell'eseguibile in funzione;
- gli eventuali argomenti passati all'eseguibile al momento dell'avvio attraverso la riga di comando;
- il numero di identificazione del processo;
- il numero di identificazione del processo che ha generato quello a cui si fa riferimento;
- il nome del dispositivo di comunicazione se il processo controllato da un terminale;
- il numero di identificazione dell’utente;
- il numero di identificazione del gruppo;

Attributi dei processi

- A ogni processo sono associati due utenti
  - **Real user:** utente che ha lanciato il processo
  - **Effective user:** utente che determina i diritti del processo
- quando il processo apre un file, vale l'effective user
- di solito, i due utenti coincidono se invece un file eseguibile ha il bit "*set user ID*" impostato, il corrispondente processo ha:
  - Real user: utente che ha lanciato il processo
  - Effective user: utente proprietario dell'eseguibile

Attributi dei processi

- Ogni processo ha anche due gruppi associati
  – real group ed effective group
- Un programma con “set user ID” è ping

```
> ls -l /bin/ping
-rwsr-xr-x 1 root root 30804 2006-10-16 19:32 /bin/ping
```

- ping ha bisogno di partire con permessi di amministratore

Processi

Il comando **ps** fornisce i processi presenti nel sistema:

```
user> ps   # fornisce i processi dell'utente associati al terminale corrente

   PID        TTY        TIME     CMD
 23228        pts/15     0:00     xdvi.bin
  9796        pts/15     0:01     bash
 23216        pts/15     0:04     xemacs-2
  9547        pts/15     0:00     csh
```

Legenda: `PID` = PID; `TTY` = terminale (virtuale); `TIME` = tempo di CPU utilizzato; `CMD` = comando che ha generato il processo.

ps [selezione] [formato]

Selezione:

- niente processi lanciati dalla shell corrente
- -u pippo i processi dell'utente pippo
- -a (All) tutti i processi

Formato:

- niente PID, terminale, ora di esecuzione, comando
- -f (full) anche UID, PPID, argomenti
- -F (Full) anche altro
- -o elenco_campi visualizza i campi specificati

ps [selezione] [formato]

- Esempi
  - ps -u $USER
  - ps -u $USER -F
  - ps -u $USER -o pid,cmd
  - ps -a -F
  - ps -a | less
  - ps -a | grep bash
  - pids=$(ps -a -o pid)

Terminazione di un processo

Per arrestare un processo in esecuzione si può utilizzare

- la sequenza Ctrl-c dal terminale stesso su cui il processo è in esecuzione;
- il comando kill seguito dal PID del processo (da qualsiasi terminale):

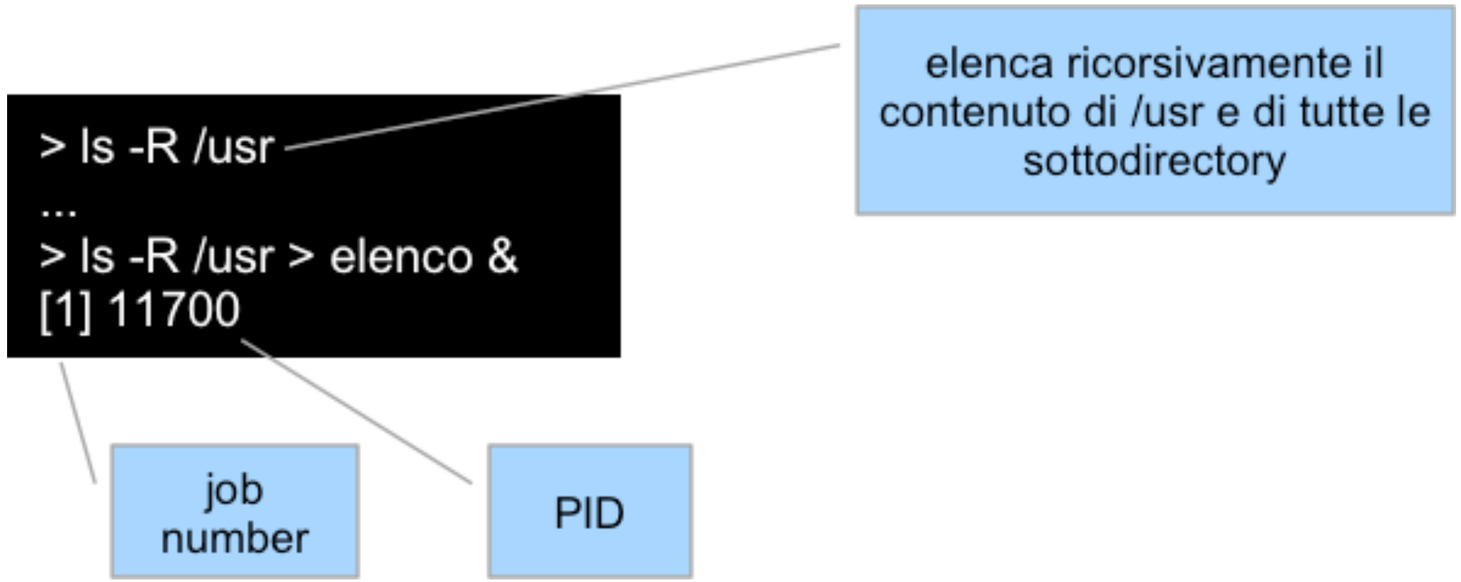
```
user> ps

  PID    TTY       TIME CMD
      .......
  28015  pts/14    0:01 xemacs
      .......

user> kill 28015
```

Controllo dei processi

- Normalmente, la shell aspetta che ogni comando termini (comando in foreground)
- Con “comando&”, la shell non aspetta (comando in background)
  - Il comando può comunque scrivere su standard output
- Esempio:



Processi in background

- I processi in background sono eseguiti in una sottoshell, in parallelo al processo padre (la shell) e non sono controllati da tastiera.
- I processi in background sono quindi utili per eseguire task in parallelo che non richiedono controllo da tastiera.

```
user> xemacs &
  [1] 24760
```

"1" il numero di job (i job sono gestiti dalla shell corrente), mentre ” 24760" il numero di processo (i processi sono gestiti dal sistema operativo).

Per terminare questo job/processo si può utilizzare sia **kill %1** che **kill 24760**.

Jobs e Processi

- Non si deve confondere un job di shell con un processo.
- Un comando impartito attraverso una shell può generare più di un processo, per esempio quando viene avviato un programma o uno script che avvia a sua volta diversi programmi, oppure quando si realizzano dei condotti.
- Un job di shell rappresenta tutti i processi che vengono generati da un comando impartito tramite la shell stessa.

Controllo dei Processi

Un job si può **sospendere** e poi **rimandare in esecuzione**

```
user> cat >temp    # job in foreground

   Ctrl-z   # sospende il job

  [1]+ Stopped

user> jobs
  [1]+ Stopped     cat >temp

user> fg    # fa il resume del job in foreground

   Ctrl-z   # sospende il job

user> bg    # fa il resume del job in background

user> kill %1   # termina il job 1
  [1]+ Terminated
```

Monitoraggio Memoria

Il comando `top` fornisce informazioni sulla memoria utilizzata dai processi, che vengono aggiornate ad intervalli di qualche secondo. I processi sono elencati secondo la quantità di tempo di CPU utilizzata.

```
user> top
load averages: 0.68,  0.39, 0.27 14:34:55
245 processes: 235  sleeping, 9 zombie, 1 on cpu
CPU states: 91.9% idle, 5.8% user, 2.4% kernel, 0.0% iowait, 0.0% swap
Memory: 768M real, 17M free, 937M swap in use, 759M swap free

  PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
12887 root       1  59    0   65M   56M sleep  105:00  3.71% Xsun
 4210 pippo      1  48    0 2856K 2312K cpu      0:00  1.50% top
 9241 root       1  59    0   35M   26M sleep   15:58  1.47% Xsun
24389 pluto      4  47    0   28M   25M sleep   16:30  0.74% opera
  ......
```

Legenda: la prima riga indica il carico del sistema nell'ultimo minuto, negli ultimi 5 minuti, negli ultimi 15 minuti, rispett.; il carico è espresso come numero di processori necessari per far girare tutti i processi a velocità massima; alla fine della prima riga c'è l'ora; la seconda contiene numero e stato dei processi nel sistema; la terza l'utilizzo della CPU; la quarta informazioni sulla memoria; le restanti righe contengono informazioni sui processi (THR=thread, RES=resident)

Riferimenti

- Capitolo 4 di [Introduction to Linux]

THANK YOU FOR YOUR ATTENTION