Sistemi Operativi[1]

Mattia Monga

Dip. di Informatica e Comunicazione
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

a.a. 2009/10

Lezione XIV: Unix power tools e primi esperimenti col kernel

find

Per selezionare file con determinate caratteristiche si usa `find`

find percorso predicato

Seleziona, nel sottoalbero definito dal percorso, tutti i file per cui il predicato è vero

Spesso usato insieme a a `xargs`

find percorso predicato| xargs comando

funzionalmente equivalente a

comando $(find percorso predicato)

ma evita i problemi di lunghezza della riga di comando perché xargs si preoccupa di "spezzarla" opportunamente.

Esercizi

1. Trovare il file piú "grosso" in un certo ramo
2. Copiare alcuni file (ad es. il cui nome segue un certo pattern) di un ramo in un altro mantenendo la gerarchia delle directory
3. Calcolare lo spazio occupato dai file di proprietà di un certo utente
4. Scrivere un comando che conta quanti file ci sono in un determinato ramo del filesystem
5. Aggiungere 10 utenti prendendo la lista da un file di testo

Archivi

Un archivio *archive* è un file di file, cioè un file che contiene i byte di diversi altri file e i relativi *metadati*. (Cfr. con una directory, che è un file speciale, che sostanzialmente contiene solo l'elenco dei file)

- `ar` L'archiviatore classico, generalmente utilizzato per le librerie (provare ar t /usr/lib/i86/libc.a
- `tar` Tape archive, standard POSIX
  tar cvf archivio.tar lista_files

Gli archivi possono essere compressi con `compress` o, piú comunemente ma assenti nel setup di MINIX di base, con `gzip` o `bzip2`
I file `.zip` sono archivi compressi.

Montare un file system

Il file system di UNIX è un albero (in realtà un DAG, perché come si vedrà ci possono essere dei link "trasversali") con una sola radice.
L'introduzione di un nuovo pezzo di file system (per esempio contenuto in un dischetto, una chiavetta USB, un disco di rete) si dice montare il file system. Occorre scegliere un punto di montaggio: il file system diventerà un sottoalbero la cui radice è tale punto (eventuali sottoalberi già esistenti non saranno piú accessibili)
mount /dev/fd0 /mnt

Altre utility

Altre utility "standard" di cui è bene conoscere almeno l'esistenza

Prog. (sez. man)	Descrizione
`od (1)`	dump files in octal and other formats
`uniq (1)`	report or omit repeated lines
`cut (1)`	remove sections from each line of files
`tr (1)`	translate or delete characters
`dd (1)`	convert and copy a file
`stat (1)`	display file or file system status
`test (1)`	check file types and compare values
`tee (1)`	read from standard input and write to standard output
`basename (1)`	strip directory and suffix from filenames
`dirname (1)`	strip non-directory suffix from file name
`sed (1)`	stream editor for filtering and transforming text

Inoltre è molto utile conoscere le espressioni regolari (`man 7 re_format`), usate da `grep`, `sed`, ecc.

Esercizi

1. Creare un archivio `tar.gz` contenente tutti i file la cui dimensione è minore di 50KB
2. Rinominare un certo numero di file: per esempio tutti i file `.png` in `.jpg`
3. Creare un file da 10MB costituito da caratteri casuali (usando `/dev/random`) e verificare se contiene la parola MINIX
4. Trovare l'utente che ha il maggior numero di file nel sistema
5. Trovare i 3 utenti che, sommando la dimensione dei loro file, occupano piu' spazio nel sistema.

Segnali

```
/* Regular signals. */
#define SIGHUP 1 /* hangup */
#define SIGINT 2 /* interrupt (DEL) */
#define SIGQUIT 3 /* quit (ASCII FS) */
#define SIGILL 4 /* illegal instruction */
#define SIGTRAP 5 /* trace trap (not reset when caught) */
#define SIGABRT 6 /* IOT instruction */
#define SIGBUS 7 /* bus error */
#define SIGFPE 8 /* floating point exception */
#define SIGKILL 9 /* kill (cannot be caught or ignored) */
#define SIGUSR1 10 /* user defined signal # 1 */
#define SIGSEGV 11 /* segmentation violation */
#define SIGUSR2 12 /* user defined signal # 2 */
#define SIGPIPE 13 /* write on a pipe with no one to read it */
#define SIGALRM 14 /* alarm clock */
#define SIGTERM 15 /* software termination signal from kill */
#define SIGEMT 16 /* EMT instruction */
#define SIGCHLD 17 /* child process terminated or stopped */
#define SIGWINCH 21 /* window size has changed */
```


Segnali

```
/* POSIX requires the following signals to be defined, even if they are
 * not supported. Here are the definitions, but they are not supported.
 */
#define SIGCONT 18 /* continue if stopped */
#define SIGSTOP 19 /* stop signal */
#define SIGTSTP 20 /* interactive stop signal */
#define SIGTTIN 22 /* background process wants to read */
#define SIGTTOU 23 /* background process wants to write */
```


Mandare segnali

- I segnali possono essere emessi col programma `kill`
- È possibile "intrappolare" i segnali ricevuti, ossia eseguire una routine di risposta

```
#!/bin/sh

trap "echo Ho ricevuto il segnale USR1" 10
trap "echo Ho ricevuto il segnale USR2" 12
trap "echo Ho ricevuto il segnale INT" 2

while true ; do
      sleep 1
      echo "ciao: sono $(id)"
done
```

- `SIGKILL` non può essere catturata.

Link

- http://www.gnu.org/software/fileutils/fileutils.html

Make

Stuart Feldman, 1977 at Bell Labs.
Permette di specificare dipendenze fra processi di generazione.
Dipendenze: se cambia questo file, allora il processo di generazione deve essere ripetuto.

```
1 helloworld: helloworld.o
2         cc -o $@ $<
3
4 helloworld.o: helloworld.c
5         cc -c -o $@ $<
6
7 .PHONY: clean
8 clean:
9         rm helloworld.o helloworld
```


Concetti fondamentali

Breakpoint

Un punto del programma in cui l'esecuzione deve essere bloccata, tipicamente per esaminare lo stato in quell'istante.

Stepping

Eseguire il programma *passo a passo*. La granularità del passo può arrivare fino all'istruzione macchina.

Stato

Lo stato del programma può essere analizzato come:

- forma simbolica: secondo i simboli definiti nel linguaggio di alto livello e conservati come *simboli di debugging*
- memoria virtuale: stream di byte suddiviso in segmenti
  - Text: contiene le istruzioni (spesso read only)
  - Initialized Data Segment: variabili globali inizializzate
  - Uninitialized Data Segment (bss): variabili globali non inizializzate
  - Stack: collezione di *stack frame* per le chiamate di procedura. Cresce verso il basso.
  - Heap: Strutture dati create dinamicamente. Cresce verso l'alto.

Uso del debugger

- `break ...`
- `run ...`
- `print ...`
- `next`
- `step`
- `backtrace`

Iniziamo a dare un'occhiata al codice

Per ricompilare il sistema

```
1 cd /usr/src/tools
2 make clean
3 make image
4 make hdboot
```

Ci mette 3-5 minuti. Esercizio: Cambiare Il nome del sistema che appare nella prima riga col copyright

Debug dumps

Premendo i tasti F1-F7, F10-F12, Shift+F1-F4 vengono stampate informazioni sulle strutture dati del kernel

- Identificare i valori del parametro di boot `memory`