Sistemi Operativi (Laboratorio)

Lorenzo Martignoni

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano, Italia
lorenzo@security.dico.unimi.it

a.a. 2008/09

Lezione IV: Shell III

Niente panico!

Materiale del corso

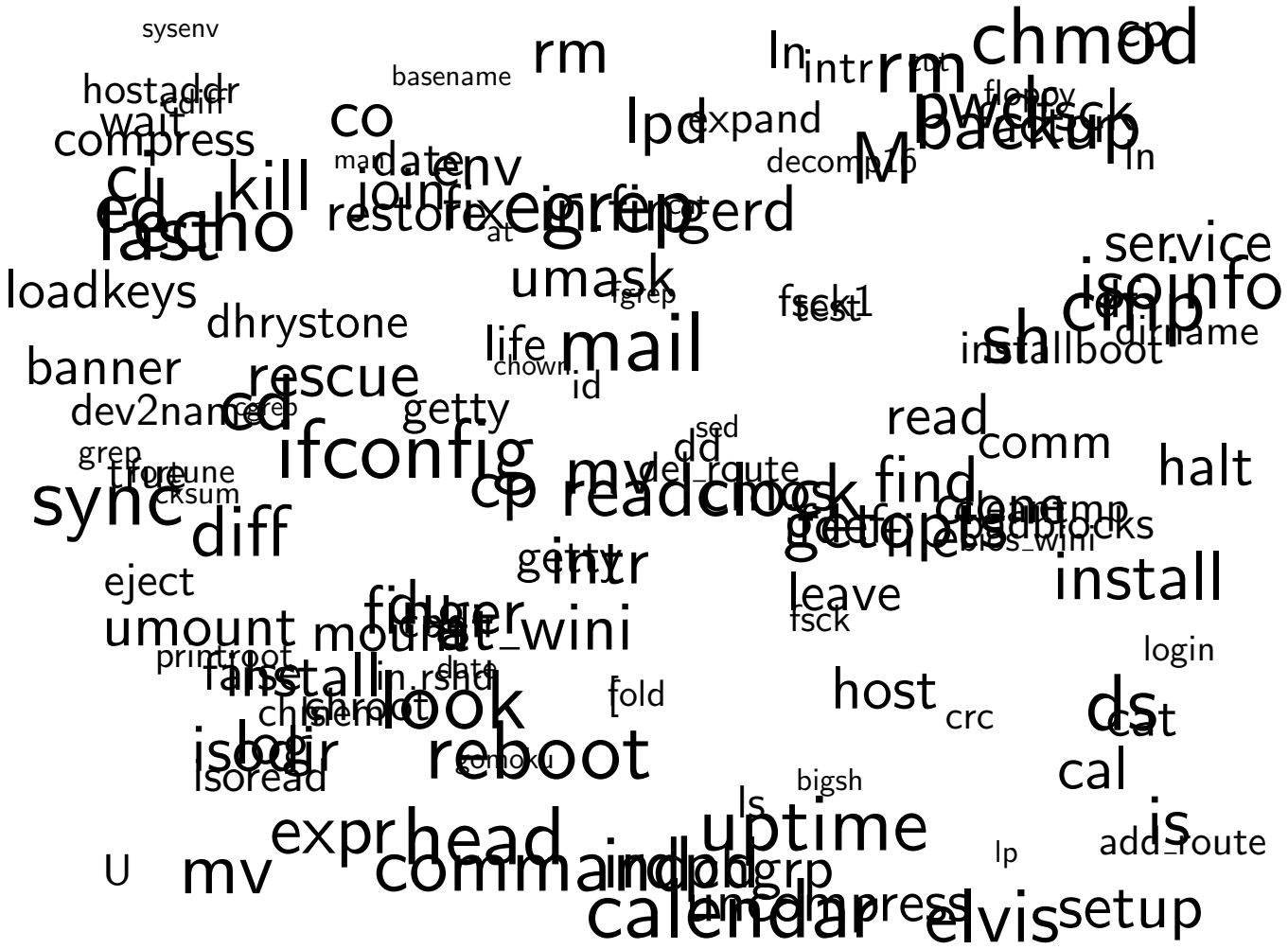
`http://homes.dico.unimi.it/~sisop/`

Tutto il materiale è sul WIKI (link nella pagina principale)

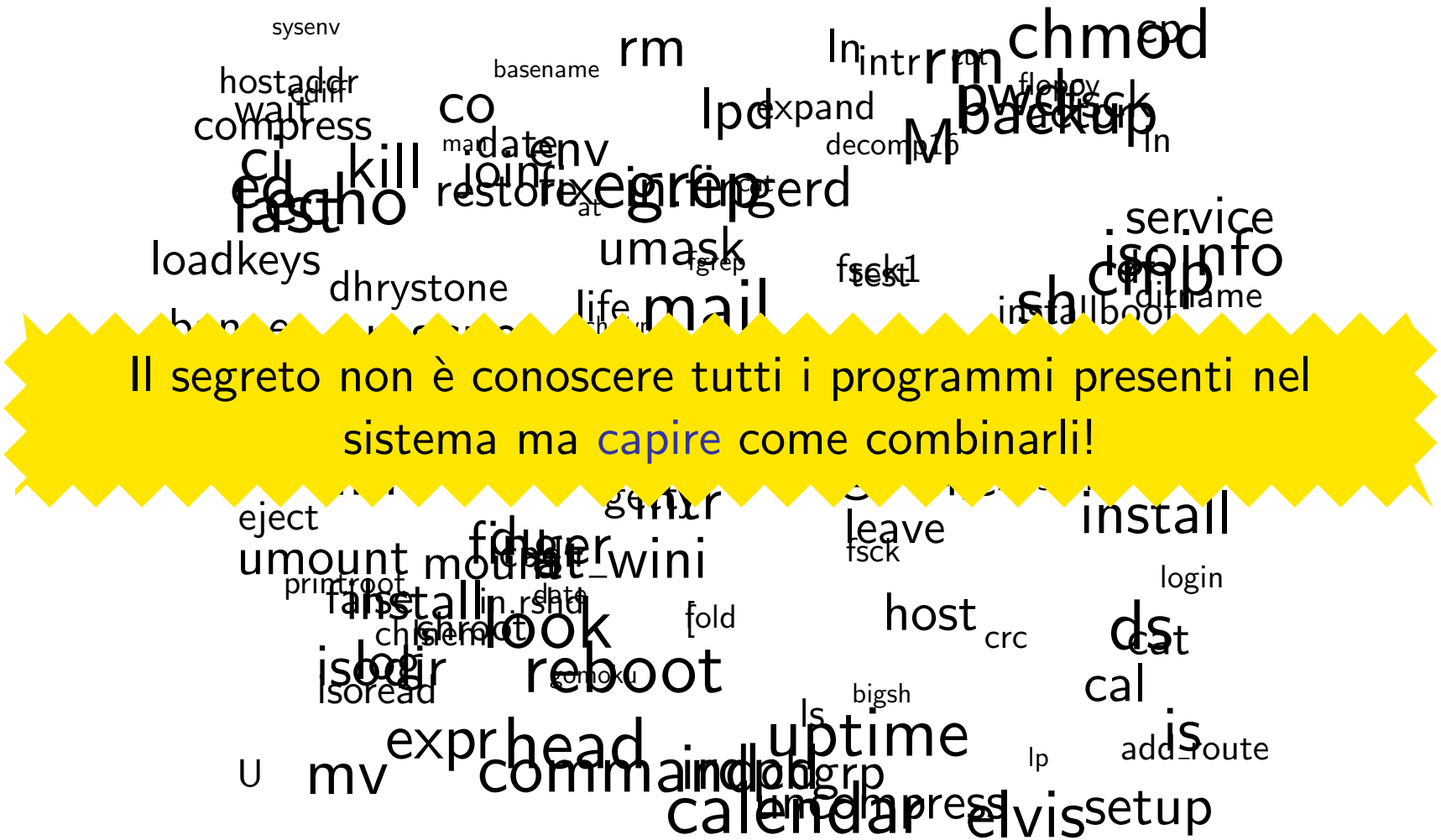
Esercizi online (link nel WIKI)

Ricevimento su appuntamento

Troppi comandi da ricordare!



Troppi comandi da ricordare!



Argomenti da riga di comando

```
1  #!/bin/sh
2
3  echo "$# argomenti"
4  a=1
5  for i in $*
6  do
7      echo "$a: $i"
8      a=$(expr $a + 1)
9  done
```

- ./script "a bb"cc dd
- a=0; ./script "a bb"cc $a
- a=0; ./script "$a"
- a=0; ./script '$a'
- a=0; ./script \$a
- ./script $(ps)

Concatenazione di programmi

- ls ; **ps** ; **echo** $?
- sleep 10 & **echo** "io non aspetto"
- cmp /tmp/file0 /tmp/file1 && **echo** "i file sono uguali"
- ls /tmp/lock && **echo** "lock impostato"
- ls /tmp/lock || **echo** "lock non impostato"
- **ps** −alx | grep "floppy"

Concatenazione di programmi

- ls ; **ps** ; **echo** $?
- sleep 10 & **echo** "io non aspetto"
- cmp /tmp/file0 /tmp/file1 && **echo** "i file sono uguali"
- ls /tmp/lock && **echo** "lock impostato"
- ls /tmp/lock || **echo** "lock non impostato"
- **ps** −alx | grep "floppy"
- (sleep 10 ; **echo** "io aspetto") & **echo** "finito"
- **echo** $(ls)

Variabili d'ambiente e non

- $?: exit status ultimo processo eseguito
- $$: PID shell
- $*: lista di argomenti ricevuti via riga di comando
- $#: numero di argomenti ricevuti via riga di comando

Variabili d'ambiente e non

- $?: exit status ultimo processo eseguito
- $$: PID shell
- $*: lista di argomenti ricevuti via riga di comando
- $#: numero di argomenti ricevuti via riga di comando

- a=0; **echo** $a
- a="sardegna"; b="sicilia"; **export** a; **echo** $a $b; sh script

#!/bin/sh

echo $a $b

If, for, while

```
if test $a -lt 10
then
    echo "$a < 10"
fi
```

If, for, while

```
if test $a -lt 10
then
    echo "$a < 10"
fi
```

```
if [ $a -lt 10 ]
then
    echo "$a < 10"
fi
```

If, for, while

```
if test $a -lt 10
then
    echo "$a < 10"
fi

a=0
while test $a -lt 100
do
    echo "$a"
    a=$(expr $a + 1)
done
```

```
if [ $a -lt 10 ]
then
    echo "$a < 10"
fi
```

If, for, while

```
if test $a -lt 10
then
    echo "$a < 10"
fi

a=0
while test $a -lt 100
do
    echo "$a"
    a=$(expr $a + 1)
done
```

```
if [ $a -lt 10 ]
then
    echo "$a < 10"
fi

u=$(cat /etc/passwd | cut -f 1 -d ":")
for i in $u
do
    echo "Utente: $i"
done
```

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 `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.

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.

Altre utility

Altre utility "standard" da conoscere

- `tee`
- `grep`
- `sed`
- `dd`

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

Segnali

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

Segnali

```
1 /* POSIX requires the following signals to be defined, even if they are
2  * not supported. Here are the definitions, but they are not supported.
3  */
4 #define SIGCONT 18 /* continue if stopped */
5 #define SIGSTOP 19 /* stop signal */
6 #define SIGTSTP 20 /* interactive stop signal */
7 #define SIGTTIN 22 /* background process wants to read */
8 #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

```
1  #!/bin/sh
2
3  trap "echo Ho ricevuto il segnale USR1" 10
4  trap "echo Ho ricevuto il segnale USR2" 12
5  trap "echo Ho ricevuto il segnale INT" 2
6
7  while true ; do
8      sleep 1
9      echo "ciao: sono $(id)"
10 done
```

- `SIGKILL` non può essere catturata.

Link

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

Esercizi I

1. Scrivere una *pipeline* di comandi che identifichi il le informazioni sul processo `/bin/floppy` (`ps`, `grep`)
2. Scrivere una *pipeline* di comandi che identifichi il solo processo che occupa piú spazio in memoria (`ps`, `sort`, `tail`)
3. Ottenere il numero totale dei file contenuti nelle directory `/usr/bin` e `/var` (`ls`, `wc`, `bc`)
4. Verificare qual è il valore di ritorno di una pipe, anche in caso che qualcuno dei "filtri" fallisca.
5. Trovare il file piú "grosso" in un certo ramo
6. Calcolare lo spazio occupato dai file modificati nell'ultima settimana
7. Scrivere un comando che conta quanti file ci sono in un determinato ramo del filesystem

Esercizi II

8. Copiare alcuni file (ad es. il cui nome segue un certo pattern) di un ramo in un altro mantenendo la gerarchia delle directory
9. Si immagini di avere un file contenente il sorgente di un programma scritto in un linguaggio di programmazione in cui i commenti occupino intere righe che iniziano con il carattere #. Scrivere una serie di comandi per ottenere il programma senza commenti. (`grep`)
10. Ottenere la somma delle occupazioni dei file delle directory `/usr/bin` e `/var`
11. Scrivere uno script `seq.sh` che riceve in input da riga di comando due interi a e b ed stampa su standard output tutti gli interi $i : a \leq i < b$
12. Creare un archivio `tar.gz` contenente tutti i file la cui dimensione è minore di 50KB

Esercizi III

13. Rinominare un certo numero di file: per esempio tutti i file `.png` in `.jpg` (`find`, `mv`)
14. Creare un file da 10MB costituito da caratteri casuali (usando `/dev/random`) e verificare se contiene la parola MINIX (`dd`)

Bibliografia

Dijkstra, E. W.
My recollections of operating systems design.
http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1303.PDF, 2001.
EWD-1303.

Tanenbaum, A., and Woodhull, A.
Operating Systems Design and Implementation, III ed.
Prentice Hall, 2006.

Copyright

Immagini tratte da [2] e da Wikipedia.