APPUNTI SISTEMI OPERATIVI

shell: interfaccia a comandi
bash: sh = shell, csh, tcsh, ksh, zsh
ambiente grafico = X: kde, gnome, unity.

comandi:
man: manuale
man qualcosa: info su qualcosa
ls: vedi cartella
cp: copy
rm: rimuovi
mv: muovi
cd: entra in dir
mkdir: crea dir
rmdir: rimuove dir
pwd: dir corrente
cd /: level up
~: home

type: mi dice come la bash considera il comando

ps: mostra i processi
ps aux: tutti i processi, anche non inerenti al terminale

pstree: mostra i collegamenti padre-figlio come un albero

top: mostra i processi in live
htop: mostra i processi colorati e con piu info

per cercare, per esempio in man:
/qualcosa

- nell'esecuzione di un comando, la **shell** crea un processo **figlio**.
- ogni processo ha una sua directory corrente

cp: esempio di comando **interno**, viene processato dalla shell stessa

cartella **usr**: unix system resources (no user)

- dopo aver scritto **ps**:
TTY: telescrivente, indica il terminale a caratteri
PID: pid del processo

/sbin/init viene lanciato al boot (processo)

- dopo aver eseguito top o htop
[kernelsthreads] tra parentesi quadre
gli altri sono i nomi dei processi

STAT: stato del processo.
S: sleeping, attesa di un evento
R: ready e running
Z: c'è stato un processo ma è gia terminato
T: stoppato (traced)

fg: foreground
nomeprocesso & lo fa parti in fg
bg: background
operano solo sulla shell corrente per processi lanciati dalla shell

CTRL+C termina processo fg
CTRL+Z stoppa processo

jobs: dice i processi controllati dalla shell
dopo posso fa fg %numero per runnarlo

kill PID: chiude processi
kill -l: segnali che posso inviare ai processi
kill -19: sigstop, stoppa
kill -18: riparte
kill -9: killa sempre

START: instante in cui è partito il processo
TIME: tempo per cui i processi sono stati attivi sulla cpu.

sudo: swtch user do, fallo da admin
apt-get install nomepacchetto

- ogni processo ha un **environment**, ambiente: coppie nome valore.

```
x=1
echo $x
$$ PID

bash
echo $$
echo $SHLVL
echo $x
exit
echo $$
echo $x
```

le variabili d'ambiente vengono esportate nelle shell piu interne
con export x

ogni processo ha **stdin, stdout, stderr**
open in c torna un intero
possiamo collegare stdout di un processo con stdin di un altro processo (pipeline)

la tastiera va in stdin, stdout e stderr tornano su monitor

con **>** si manda in stdout
con **<** si prende dallo stdin

cat z1 z2 z3 > z4
z4 è la concatenazione di z123

si posso creare **filtri**
tr fa lo string replace
esempio tr : -
sostituisce i due punti in un trattino
esempio tr p x
sostituisce le p con le x

dovrebbe mettere dentro un file z il contenuto del file passwd sostituendo : con -
cat /etc/passwd | tr : - > z

cat /etc/passwd | sort | less
less: visualizzatore di testo nella shell
sort: ordina lessicograficamente

cat /etc/passwd | sort | **head**
torna le **prime dieci righe**

cat /etc/passwd | sort | head **-n1**
mostra solo la **prima riga**

cat /etc/passwd | sort | tail -n +2 | less
leva le prime 2 righe

ps aux | tail -n +2 | less

ps aux | tail -n +2 | wc
ritorna **num righe, num parole, num chars**
wc -l ritorna solo num righe

ps aux **-no-headers** | less
leva la prima riga credo (provare!)

al post di head metto **tail** e fa la stessa cosa ma tail mostra le **ultime** righe

tee nomefile crea un file in cui dentro ci metto il contenuto del comando dato, tipo quelli sopra

esiste un filtro chiamato **cut** che permette di **isolare** dei campi
cut /etc/passwd | cut -d : -f 3
stampa il campo 3
-d: delimitatore di campo, in questo caso i due punti :

grep xxx seleziona le righe
egrep ":1*0"
denota un match con :1...1...1..0
un certo numero di uni con 0 finale

| uniq rimuove le righe consecutive identiche
sort | uniq rimuove tutte le righe doppioni
uniq -c | sort -n -r
boh (provare!)

Matching, espressioni regolari:
[a-zA-Z] lettere qualsiasi
[02468] numeri pari
[0-9] numeri qualsiasi
a(b|c)d a seguita da b or c, seguita da d

operatore di iterazione
ciao: ____ 1
per matchare n volte un char si mette ***
ab*c
la b viene matchata n volte
a{5} matcha 5 volte a
. il punto matcha **un** carattere qualsiasi
.* matcha n caratteri qualsiasi
ciao.*bella tra ciao e bella ci puo sta qualsiasi cosa
\. \\ * matchano punto, slash e asterisco
[^a] matcha tutto tranne la a
grep -i è case insensitive
[^:]*: matcha tutte le cose tra i due punti
$ matcha la fine riga
^ matcha inizio riga
^blabla$ matcha solo blabla

matcho le righe che iniziano per p
cat /etc/passwd | **grep ^p**
per farlo capire alla shell si mette tra apici singoli: **grep '^p'**
grep -color=auto always never
grep '**^p[^:]*:x:[0-9]*[13579]:**'
matcha terzo campo che finisce per cifra dispari
grep -v inverte il matching

'**^[^:]*:x:[0-9]*[13579]:**'

'**^[^:]*[^:]*:[0-9]*[13579]:**'

esercizio: tutte le righe che hanno come ultimo campo /**false** oppure **/nologin**

egrep '^[^:]*:[^:]*:[^:]*:[^:]*:[^:]*:[^:]*:[^:]*:[^:]*:/bin/(false|nologin)
egrep ':.*/(false|nologin)$'

linguaggio awk

pattern { statement }
per tutti i pattern esegue gli statement
senza statement fa la stampa

cat /etc/passwd | awk -v FS=: '/:.*\/(false|nologin)$/'
definisco un nuovo delimitatore di campo e lo metto uguale a due punti, poi matcho i campi che finiscono con false o nologin
FS: feed separator
/ inizio e fine espressione sono delimitati da **/**
***\/** per fare la slash e non fargliela interpretare come fine riga

-v serve per inizializzare le variabili: tipo x=9

cat /etc/passwd | awk -v FS=: '/:.*\/(false|nologin)$/ { print "ciao" }'

stampa ciao per ogni riga che matcha

cat /etc/passwd | awk -v FS=: '/:.*\/(false|nologin)$/ { n=n+1; print "ciao " n }'

stampa ciao per ogni riga che matcha e la variabile n inizia da 0 e si incrementa ad ogni riga

"1" + "2" concatenazione di stringhe
awk non ha tipi ma solo stringhe

-v n=10 inizializzo la variabile n a 10

{print "**" $0 "**" }

| **wc** conta le righe (primo valore che da)
wc -l fa vedere solo il numero righe

tutte le righe che iniziano per p awk **'/^p/'**
oppure al post di awk metto grep che ha i colori

nelle espressioni regolari posso usare:
– tutti gli operatori di confronto e matematici
– && || !

{n=n+1; print n " " $0 }
numera ogni riga

stampa solo se n<10
n<10 {n=n+1; print n " " $0 }

tutte le righe che **non** iniziano per p: **'!/^p/'**

{n=n+1} !{n<10} {print n " " $0 }

oppure uso NR (number of records) mi numera tutte le righe
awk ' (NR < 10) {print NR " " $0 } '
stampa le righe fino a 9

awk divide per campi, $1, $2, ecc
viene fatta tramite un divisore di campo, di default è lo spazio o tabulazione.
FS (field separator) posso separare tramite un delimitatore personalizzato
inizializza la var di separator a due punti
awk -v FS=:
awk -v FS=: '{print $1,$7}'

tutti quelli per cui il settimo campo non è bin/false
awk -v FS=: '$7!="/bin/false" {print $1,$7}'
la virgola equivale ad uno spazio

NF (number of fields) mi dice il numero di campi che sono stati trovati nel record.
$5 quinto campo
$NF mi da l'ultimo campo
$(NF-1) mi da il penultimo
funge da indice

{for(i=1; i<=NF; i++) {printf $(i)}

stampa le righe che contengono nel primo campo dave
awk -v FS=: '$1=="dave" {for(i=1; i<=NF; i++) {printf $(i)} '

{if ($1 == "dave" } {for … { printf"%s:",$(i) }; printf "\n" } }

sostituisce x con la stringa ciao:
awk -v FS=: '{$2="ciao"; printf $0}'
il separatore diventa spazio
quindi devo mettere -v OFS=:
che sarebbe output feed separator

awk '**BEGIN** {FS=":"; OFS=":" } …
viene eseguito **prima di qualsiasi cosa**
viene usato per inizializzare una variabile
'BEGIN {FS=":"; OFS=":"; s=0} {print $0; s+=$3} END {print s}'
END lo fa per ultimo

sostituire una parola con un'altra
awk '{gsub(/^dave:/, "abcde:"); } {print $0}'

oppure
$1==pizzonia {$1=abcde} {print $0}

esercizio:
sostituire nell'ultimo campo tutte le / con trattini
awk 'BEGIN {FS=":"; OFS=":"} {gsub(/\//,"-", $NF); print $0}'

gsub: fa sostituzione di stringhe

RS (record separator) delimitatore di record (non campi)
-v RS="" -v FS="\n" '{print $1}'

echo: stampa messaggi sulla shell
echo a b c a b c
echo a{1,2,3} a1 a2 a3
echo a{1,2,3}b a1b a2b a3b

fa il prodotto cartesiano
echo a{1,2,3}b{4,5} a1b4 a1b5 a2b4... a3b5

per mettere lo spazio si mette **"\ "**
echo a{1,\ ,3}b{4,5} a1b4 a1b5 a b4 a b5…

stampa la home dello user
echo **~** /home/dave
echo ~root /root

echo a b $x ~root a{1,2}
a b /home/dave /root a1 a2

assegnazione di variabili
x=~root
echo $x /root
x=~
echo $x /home/dave

assegno ad x un comando
x="ls -l"
richiamo x e quindi mi esegue ls -l
$x percorso

espressioni aritmetiche $((…))
echo $((3 + 5)) 8

scrivo dentro ad un file a b c e lo creo
echo a b c > nomefile

mostro il contenuto del file con cat
cat nomefile

stampa contenuto file con echo
echo `cat nomefile` a b c

touch nomefile: crea un file nomefile
rm nomefile: rimuove il file nomefile

crea tre file chiamati a b c perchè dentro nomefile prima ho messo a b c:
touch `cat nomefile`

questo invece crea due file chiamati uno cat e l'altro nomefile: **touch** cat nomefile
in quanto non ho messo gli apici strambi

System programming (#C)

Direttive:
include, define, ifdef

.c cpp→ .i cc→ .s as→ .o esegui

Linker: mette insieme più file object.

Compilatore gnu:
gcc -o hello hello.c
crea un po di file .s e .o

file nomefile mi dice info sul file

linkaggio statico: funzione dentro eseguibile
vantaggio: eseguibile dipende solo dal kernel
svantaggio: eseguibile grosso e occupa tanta memoria
-static

linkaggio dinamico: codice condiviso tra tanti eseguibili, tipo la printf.
vantaggio: occupa poco spazio in memoria, se l'aggiornate, non si deve aggiornare l'exe
svantaggio: mi porto dietro le librerie dinamiche

.so .dll: librerie dinamiche
so = shared object
dll = dynamic link library

libc è la libreria che ha gran parte delle funzioni standard di c (tipo printf)
libm contiene le funzioni matematiche

ldd /bin/ls mi dice quali sono le lib dinamiche con cui l'eseguibile è linkato.

ldd ./hello
vedo che librerie usa il file hello:
appare: libc.so
/lib/ld-linux.so

linker dinamico: mappa in memoria l'eseguibile e le lib dinamiche e crea collegamenti. Si trova nell'immagine del processo.

gcc -o hello hello.c -static
file hello poi mi dice che è staticamente linked
ldd hello mi dice not dynamic exe

fermare compilazione:
gcc hello.c -E
-E si ferma alla precompilazione e la stampa

gcc hello.c -S -o hello.s
-S si ferma all'assembly e lo stampa

gcc hello.c -c -o hello.o
-c si ferma al file oggetto e genera hello.o
mancano gli indirizzi delle istruzioni da chiamare

objdump -h hello.o
fa vedere informazioni sui file oggetto
ELF: file format
.text contiene il codice
.data i dati, variabili globali
.rodata read only data

objdump -s hello.o | less
objdump -D hello.o | less
objdump -L hello.o | less

objdump -t hello.o | less
tabella dei simboli di un file object

gcc -o hello hello.o
objdump -h hello

objdump -T hello
dice che l'eseguibile ha un simbolo indefinito chiamato printf che si aspetta di trovare in qualche libreria dinamica.

Il debugger decodifica l'immagine di processo con le variabili locali.

gdb: gnu debugger, non ha interfaccia grafica
run avvia il programma
segmentation fault: processo ha acceduto ad una ragione non ammessa (non perchè manca il segmento)
l: 10 righe intorno all'errore?
p: print
b: break numerolinea
next: riga successiva

info br mi dice il breakpoint creato con b
p a[i]
bt
up

s: stepinto