Ulteriori comandi sui file

- Confronto tra file:

  1. `> cmp file1 file2`
     restituisce il primo byte ed il numero di linea in cui `file1` e `file2` differiscono (se sono uguali, non viene stampato nulla a video).
  2. `> diff file1 file2`
     restituisce la lista di cambiamenti da apportare a `file1` per renderlo uguale a `file2`.

- Ricerca di file:
  `> find` *<pathnames>* *<expression>*
  attraversa ricorsivamente le directory specificate in *<pathnames>* applicando le regole specificate in *<expression>* a tutti i file e sottodirectory trovati. *<expression>* può essere una fra le seguenti:
  1. opzione,
  2. condizione,
  3. azione.

Esempi d'uso di `find`

- `> find . -name '*.c' -print`
  cerca ricorsivamente a partire dalla directory corrente tutti i file con estensione `c` e li stampa a video.

- `> find . -name '*.bak' -ls -exec rm {} \;`
  cerca ricorsivamente a partire dalla directory corrente tutti i file con estensione `bak`, li stampa a video con i relativi attributi (`-ls`) e li cancella (`-exec rm {} \;` Il carattere \ serve per fare il "quote" del `;`).

- `> find /etc -type d -print`
  cerca ricorsivamente a partire dalla directory `/etc` tutte e solo le sottodirectory, stampandole a video.

Controllo di processi

Ogni processo del sistema ha un PID (Process Identity Number).Ogni processo può generare nuovi processi (figli). La radice della gerarchia di processi è il processo init con PID=1. `init` è il primo processo che parte al boot di sistema (`pstree`).

Il comando `ps` fornisce i processi dell'utente associati al terminale corrente:

```
user> ps

   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.

Per ottenere il nome del terminale corrente:

```
user> tty
 /dev/pts/15
```

Il comando `ps` e sue varianti, I

Per ottenere tutti i processi nel sistema associati ad un terminale (`-a`), full listing (`-f`):

```
user> ps -af

   UID   PID  PPID  C     STIME  TTY       TIME CMD
 pippo 10922  9560  0    Oct 17  pts/17    0:00 bash
 pluto 23410 23409  0  11:07:08  pts/26    0:01 xdvi.bin -name xdvi
  root 24188  9807  0  12:34:10  pts/13    0:00 ps -af
        ....
```

Legenda: `UID` = User Identifier; `PPID` = Parent PID; `C` = informazione obsoleta sullo scheduling; `STIME` = data di inizio del processo.

Il comando `ps` e sue varianti, II

Per ottenere tutti i processi nel sistema, anche non associati ad un terminale (`-e`), long listing (`-l`):

```
user> ps -el

 F S   UID   PID  PPID  C PRI NI   ADDR     SZ    WCHAN  TTY      TIME CMD
19 T     0     0     0  0   0 SY      ?      0           ?        0:12 sched
 8 S     0     1     0  0  41 20      ?    100        ?  ?        0:03 init
 8 S   140 12999 12997  0  56 20      ?    278        ?  pts/12   0:00 tcsh
 8 R   159  9563  9446  0  50 20      ?   2110           ?        0:30 acroread
        ....
```

Legenda: `F` = flag obsoleti; `S` = stato del processo (T=stopped); `PRI` = priorità; `NI` = nice value (usato per modificare la priorità); `ADDR` = indirizzo in memoria; `SZ` = memoria virtuale usata; `WCHAN` = evento su cui il processo è sleeping.

Terminazione di un processo

Per arrestare un processo in esecuzione si può utilizzare

- la sequenza `Ctrl-k` 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
```

- il comando `kill` con il segnale SIGKILL

```
user> kill -9 28015

user> kill -s kill 28015
```

Processi in background

Un comando (pipeline, sequenza) seguito da `&` dà luogo ad uno o più 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] è il numero del job, 24760 il PID del processo

```
user> xemacs &
user> ls -R / >temp 2>err &
```

Il comando `jobs` mostra la lista dei job in esecuzione:

```
user> jobs
  [1]   Running                 xemacs &
  [2]-  Running                 xemacs &
  [3]+  Running                 ls -R / >tmp 2>err
```

Controllo di job

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
```

Bash: priorità con nice e renice

- Il sistema operativo gestisce i processi attivi mediante un meccanismo dinamico di priorità: ogni processo possiede una priorit di Base che viene modificata runtime dal sistema oparativo in funzione dello stato del processo.

- La priorità può essere aggiustata da nice nel campo di variazione tra -20 (la più alta priorità) a +19 (la più bassa). L'utente (non amministratore) può solo diminuire la priorità dei propri processi (aumentando il nice).

- Esercizio: `(xterm&) ; (nice xterm&) ; (nice -n 19 xterm&); ps -el`

- L'utente può modificare il nice di un proprio processo in esecuzione con il comando renice.

- Esempio: `renice 19 10618` (10618 è il pid)

Monitoraggio della 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)

I comandi filtro

I *filtri* sono una particolare classe di comandi che possiedono i seguenti requisiti:

- prendono l'input dallo standard input device,
- effettuano delle operazioni sull'input ricevuto,
- inviano il risultato delle operazioni allo standard output device.

Tali comandi risultano quindi degli ottimi strumenti per costruire pipeline che svolgano compiti complessi.

Ad esempio:

```
> uniq file
```

restituisce in output il contenuto del file `file`, sostituendo le linee adiacenti uguali con un'unica linea.

Comandi filtro: `grep`, `fgrep`, `egrep`

I comandi:

- `grep`: General Regular Expression Parser,
- `fgrep`: Fixed General Regular Expression Parser,
- `egrep`: Extended General Regular Expression Parser,

restituiscono solo le linee dell'input fornito che contengono un pattern specificato tramite espressione regolare o stringa fissata.

Sintassi:

`grep` [options] pattern [filename...]
`fgrep` [options] string [filename...]
`egrep` [options] pattern [filename...]

Opzioni:

`-i`: ignora la distinzione fra lettere maiuscole e minuscole,
`-l`: fornisce la lista dei file che contengono il pattern/string,
`-n`: le linee in output sono precedute dal numero di linea,
`-v`: vengono restituite solo le linee che non contengono il pattern/string,
`-w`: vengono restituite solo le linee che contengono il pattern/string come parola completa,
`-x`: vengono restituite solo le linee che coincidono esattamente con pattern/string.

I metacaratteri delle espressioni regolari

metacarattere	tipo	significato
`^`	B	inizio della linea
`$`	B	fine della linea
`\<`	B	inzio di una parola
`\>`	B	fine di una parola
`.`	B	un singolo carattere (qualsiasi)
`[str]`	B	un qualunque carattere in `str`
`[^str]`	B	un qualunque carattere non in `str`
`[a-z]`	B	un qualunque carattere tra `a` e `z`
`\`	B	inibisce l'interpretazione del metacarattere che segue
`*`	B	zero o più ripetizioni dell'elemento precedente
`+`	E	una o più ripetizioni dell'elemento precedente
`?`	E	zero od una ripetizione dell'elemento precedente
`{j,k}`	E	un numero di ripetizioni compreso tra `j` e `k` dell'elemento precedente
`s|t`	E	l'elemento `s` oppure l'elemento `t`
`(exp)`	E	raggruppamento di `exp` come singolo elemento

dove B (basic) indica che la sequenza di caratteri è utilizzabile sia in `grep` che in `egrep`, mentre E (extended) indica che la sequenza di caratteri è utilizzabile solo in `egrep` (o in `grep` usando l'opzione `-E`).

Esempi d'uso di `grep`, `fgrep`, `egrep`

- `> fgrep rossi /etc/passwd`
  fornisce in output le linee del file `/etc/passwd` che contengono la stringa fissata `rossi`.

- `> egrep -nv '[agt]+' relazione.txt`
  fornisce in output le linee del file `relazione.txt` che non contengono stringhe composte dai caratteri `a`, `g`, `t` (ogni linea è preceduta dal suo numero).

- `> grep -w print *.c`
  fornisce in output le linee di tutti i file con estensione `c` che contengono la parola intera `print`.

- `> ls -al . | grep '^d.......w.'`
  fornisce in output le sottodirectory della directory corrente che sono modificabili dagli utenti ordinari.

- `> egrep '[a-c]+z' doc.txt`
  fornisce in output le linee del file `doc.txt` che contengono una stringa che ha un prefisso di lunghezza non nulla, costituito solo da lettere `a`, `b`, `c`, seguito da una `z`.

Comandi filtro: `sort`

Il comando `sort` prende in input delle linee di testo, le ordina (tenendo conto delle opzioni specificate dall'utente) e le invia in output.

- `sort` tratta ogni linea come una collezione di vari campi separati da delimitatori (default: spazi, tab ecc.).

- l'ordinamento di default avviene in base al primo campo ed è alfabetico.

Il comportamento di default si può cambiare tramite le opzioni:

`-b`	ignora eventuali spazi presenti nelle chiavi di ordinamento,
`-f`	ignora le distinzioni fra lettere maiuscole e minuscole,
`-n`	considera numerica (invece che testuale) la chiave di ordinamento
`-r`	ordina in modo decrescente,
`-o` *file*	invia l'output al file *file* invece che allo standard output,
`-t` *s*	usa *s* come separatore di campo,
`-k` *s1*,*s2*	usa i campi da *s1* a *s2*-1 come chiavi di ordinamento.

Esempi d'uso di `sort`

Per ordinare le righe del file `/etc/passwd` in base al terzo campo (user ID), il comando

```
> sort -t: -k3,4 /etc/passwd
 root:x:0:1:Super-User:/:/sbin/sh
 guest:x:1001:120:Guest User:/home/guest:/usr/local/bin/bash
 daemon:x:1:1::/:
 ...
```

non dà il risultato voluto in quanto di default l'ordinamento è alfabetico, mentre il campo user ID è un numero; quindi si rende necessaria l'opzione `-n`:

```
> sort -t: -k3,4 -n /etc/passwd
 root:x:0:1:Super-User:/:/sbin/sh
 daemon:x:1:1::/:bin:x:2:2::/usr/bin: ...
 guest:x:1001:120:Guest User:/home/guest:/usr/local/bin/bash ...
```

Si noti che in entrambi gli esempi il separatore (`:`) è stato impostato con l'opzione `-t:`.

Comandi filtro: `tr`

Character translation: `tr` è un semplice comando che permette di eseguire operazioni come la conversione di lettere minuscole in maiuscole, cancellazione della punteggiatura ecc. Siccome può prendere input soltanto dallo standard input e stampare soltanto sullo standard output, bisogna usare delle pipe o delle ridirezioni di input/output per farlo leggere/scrivere su file.

Sintassi di base: `> tr str1 str2`

(i caratteri in `str1` vengono sostituiti con i caratteri in posizione corrispondente della stringa `str2`)

Esempi:

- `> tr a-z A-Z`
  converte le minuscole in maiuscole.
- `> tr -c A-Za-z0-9 ' '`
  sostituisce tutti i caratteri non (opzione `-c`: complemento) alfanumerici con degli spazi.
- `> tr -cs A-Za-z0-9 ' '`
  come nell'esempio precedente, ma comprime gli spazi adiacenti in un'unico spazio (opzione `-s`: *squeeze*).
- `> tr -d str`
  cancella i caratteri contenuti nella stringa `str`.

Esercizi

- Qual è l'effetto del comando `sort file >file`, dove `file` è il nome di un file?
- Fare alcuni esperimenti per scoprire qual è l'effetto del comando `tr str1 str2` se le stringhe `str1` e `str2` hanno lunghezze diverse.
- Scrivere un comando per sostituire tutti i caratteri alfanumerici nell'input con un carattere <`Tab`>, in modo che non compaiano più <`Tab`> consecutivi.
- Scrivere una pipeline che permetta di scoprire se ci sono linee ripetute in un file.
- Visualizzare su standard output, senza ripetizioni, lo user ID di tutti gli utenti che hanno almeno un processo attivo nel sistema.

Comandi filtro: `sed` (Stream EDitor)

Permette di editare il testo passato da un comando ad un altro in una pipeline. Ciò è molto utile perché tale testo non risiede fisicamente in un file su disco e quindi non è editabile con un editor tradizionale (e.g. `vi`). Tuttavia `sed` può anche prendere in input dei file, quindi la sua sintassi è la seguente:

`sed actions files`

- Se non si specificano azioni, `sed` stampa sullo standard output le linee in input, lasciandole inalterate.

- Se vi è più di un'azione, esse possono essere specificate sulla riga di comando precedendo ognuna con l'opzione `-e`,

  oppure possono essere lette da un file esterno specificato sulla linea di comando con l'opzione `-f`.

- Se non viene specificato un *indirizzo* o un intervallo di indirizzi di linea su cui eseguire l'azione, quest'ultima viene applicata a tutte le linee in input. Gli indirizzi di linea si possono specificare come numeri o espressioni regolari.

Esempi d’uso di `sed`

- `> sed ’4,$d’ /etc/passwd`
  stampa a video soltanto le prime 3 righe del file `/etc/passwd`: `d` è il comando di cancellazione che elimina dall’output tutte le righe a partire dalla quarta (`$` sta per l’ultima riga del file).

- `> sed 3q /etc/passwd`
  stesso effetto del precedente comando: in questo caso `sed` esce dopo aver elaborato la terza riga (`3q`).

- `> sed /sh/y/:0/_%/ /etc/passwd`
  sostituisce in tutte le righe che contengono la stringa `sh` il carattere `:` con il carattere `_` ed il carattere `0` con il carattere `%`.

- `> sed ’/sh/!y/:0/_%/’ /etc/passwd`
  sostituisce in tutte le righe che *non* contengono la stringa `sh` il carattere `:` con il carattere `_` ed il carattere `0` con il carattere `%`. Si noti l’uso del quoting per impedire che la shell interpreti il metacarattere `!`.

Sostituzione del testo con `sed`

Il formato dell'azione di sostituzione in `sed` è il seguente:

`s/expr/new/flags`

dove:

- `expr` è l'espressione da cercare,
- `new` è la stringa da sostituire al posto di `expr`,
- `flags` è uno degli elementi seguenti:
  - *num*: un numero da 0 a 9 che specifica quale occorrenza di `expr` deve essere sostituita (di default è la prima),
  - `g`: ogni occorrenza di `expr` viene sostituita,
  - `p`: la linea corrente viene stampata sullo standard output nel caso vi sia stata una sostituzione,
  - `w` *file*: la linea corrente viene accodata nel file *file* nel caso vi sia stata una sostituzione.

Esempi di sostituzioni con `sed`

- `sed '/^root/,/^bin/s/:.............:/::/w disabled.txt' /etc/passwd`
  sostituisce la password criptata (lunga 13 caratteri) con la stringa vuota nelle righe in input comprese fra quella che inizia con `root` e quella che inizia con `bin`; tali righe sono poi accodate nel file `disabled.txt`.

- `cat /etc/passwd | sed 's?/bin/.*sh$?/usr/local&?'`
  cerca tutte le righe in input in cui compare la stringa corrispondente all'espressione regolare `/bin/.*sh$` (ad esempio `/bin/bash`) e sostituisce quest'ultima con la stringa corrispondente a `/usr/local/bin/.*sh$` (ad esempio `/usr/local/bin/bash`). Si noti che, siccome il carattere separatore di `sed` compare nella stringa da cercare, si è usato il carattere `?` come separatore. Inoltre il carattere `&` viene rimpiazzato automaticamente da `sed` con la stringa cercata (corrispondente a `/bin/.*sh$`).