2021_12_03

December 5, 2021

1 2021-12-03

1.0.1 Corso ITS

1.1 Magento & e-commerce software

1.2 ### Fondamenti di Programmazione (Andrea Ribuoli)

1.3 GNU make

[13]:
```
!git clone https://github.com/lucatinti13/esercizic.git; \
 cd esercizic; \
 make
```

```
Cloning into 'esercizic'…
remote: Enumerating objects: 71, done.
remote: Counting objects: 100% (71/71), done.
remote: Compressing objects: 100% (69/69), done.
remote: Total 71 (delta 32), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (71/71), 423.87 KiB | 1.69 MiB/s, done.
Resolving deltas: 100% (32/32), done.
gcc -c -o elabora.o elabora.c
gcc -o elabora elabora.o
```

Se vogliamo parametrizzare i comandi eseguiti dal **make** possiamo utilizzare le variabili d'ambiente. Sostituiamo la regola `cp elabora ..` con `cp elabora $(PERCORSO)` eseguendola cosi:

[24]:
```
!cd esercizic; \
 PERCORSO=.. make install
```

```
cp elabora ..
```

Il comando **make** accetta come argomento un *target* (obiettivo) che viene ricercato nel file avente nome **Makefile** nella cartella corrente

[27]:
```
!cd esercizic; \
 make pippo
```

```
make: *** No rule to make target 'pippo'.  Stop.
```

[15]:
```
!cd esercizic; \
 make
```

```
make: 'elabora' is up to date.
```

[17]:
```
!make --help
```

```
Usage: make [options] [target] …
Options:
  -b, -m                      Ignored for compatibility.
  -B, --always-make           Unconditionally make all targets.
  -C DIRECTORY, --directory=DIRECTORY
                              Change to DIRECTORY before doing anything.
  -d                          Print lots of debugging information.
  --debug[=FLAGS]             Print various types of debugging information.
  -e, --environment-overrides
                              Environment variables override makefiles.
  --eval=STRING               Evaluate STRING as a makefile statement.
  -f FILE, --file=FILE, --makefile=FILE
                              Read FILE as a makefile.
  -h, --help                  Print this message and exit.
  -i, --ignore-errors         Ignore errors from recipes.
  -I DIRECTORY, --include-dir=DIRECTORY
                              Search DIRECTORY for included makefiles.
  -j [N], --jobs[=N]          Allow N jobs at once; infinite jobs with no arg.
  -k, --keep-going            Keep going when some targets can't be made.
  -l [N], --load-average[=N], --max-load[=N]
                              Don't start multiple jobs unless load is below N.
  -L, --check-symlink-times   Use the latest mtime between symlinks and target.
  -n, --just-print, --dry-run, --recon
                              Don't actually run any recipe; just print them.
  -o FILE, --old-file=FILE, --assume-old=FILE
                              Consider FILE to be very old and don't remake it.
  -O[TYPE], --output-sync[=TYPE]
                              Synchronize output of parallel jobs by TYPE.
  -p, --print-data-base       Print make's internal database.
  -q, --question              Run no recipe; exit status says if up to date.
  -r, --no-builtin-rules      Disable the built-in implicit rules.
  -R, --no-builtin-variables  Disable the built-in variable settings.
  -s, --silent, --quiet       Don't echo recipes.
  -S, --no-keep-going, --stop
                              Turns off -k.
  -t, --touch                 Touch targets instead of remaking them.
  --trace                     Print tracing information.
  -v, --version               Print the version number of make and exit.
  -w, --print-directory       Print the current directory.
  --no-print-directory        Turn off -w, even if it was turned on implicitly.
  -W FILE, --what-if=FILE, --new-file=FILE, --assume-new=FILE
                              Consider FILE to be infinitely new.
  --warn-undefined-variables  Warn when an undefined variable is referenced.
```

```
This program built for powerpc-ibm-os400
Report bugs to <bug-make@gnu.org>
```

Il seguente comando prepara l'input per il programma che abbiamo predisposto

[28]:
```
!ls -b1 /usr/bin/* > tutti_i_miei_comandi.txt
```

[ ]:
```
!./elabora tutti_i_miei_comandi.txt
```

1.4 rinominare il vostro repository in EserciziC

completare la tabella coi nomi utente GitHub

2 È ora di un vero esercizio!

Ho scritto un articolo per un giornale. Ho usato il formato **solo testo**. L'articolo è stato accettato! Per stamparlo mi viene però richiesto di ri-organizzarlo in righe di non più di 30 caratteri affinchè possa essere inserito come colonna in una pagina della prossima uscita del giornale.

Scrivi un programma che accetti due parametri:

- il nome del file contenente l'articolo originale
- il numero massimo di caratteri per riga

Il programma deve emettere sullo **standard output** le righe riorganizzate secondo quanto richiesto e restituire, come valore di ritorno, il numero totale delle stesse.

[ ]:
```
!git add . ;\
 git commit -m "aggiornamento"  ;\
 git push
```

123456789012345678901234567890 ============================== Ho scritto un articolo per un giornale. Ho usato il formato solo testo. L'articolo è stato accettato! Per stamparlo mi vi ene però richiesto di ri-organ izzarlo in righe di non più di 30 caratteri affinchè possa es sere inserito come colonna in una pagina della prossima usci ta del giornale.

[75]:
```
!gcc -g -o articolo articolo.c
```

[ ]:
```
' ' = code point 32 (decimale) = x'20' (esadecimale)

|  C  | C|   C    |
|xx|__|xx|xx|__|__|xx|__|__|__|


|0.......| ascii
|110.....|10......|
|1110....|10......|10......|
|11110...|10......|10......|10......|
```

[1]:
```
!./articolo articolo.txt 30
```

Ho scritto un articolo per un giornale. Ho usato il formato solo testo. L'articolo è stato accettato! Per stamparlo mi viene però richiesto di ri-organizzarlo in righe di non più di 30 caratteri affinchè possa essere inserito come colonna in una pagina della prossima uscita del giornale. Scrivi un programma che accetti due parametri: il nome del file contenente l'articolo originale il numero massimo di caratteri per riga Il programma deve emettere sullo standard output le righe riorganizzate secondo quanto richiesto e restituire, come valore di ritorno, il numero totale delle stesse

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int dimensione;
    int lunghezza_articolo;
    char riga[256];
    char articolo_base[5000];
    char *pCh = articolo_base;
    FILE *file;
    int x=0;
    if(argc<2) {
        printf("Non ci sono abbastanza parametri, controlla di aver messo il nome del file\n")
        exit(-1);
    }
    if(argc<3) {
        printf("Non ci sono abbastanza parametri, controlla di aver messo il numero massimo di
        exit(-2);
    }
    dimensione = atoi(argv[2]);
    file=fopen(argv[1],"r");
    if(file==NULL)
    {
```

```
        printf("il file %s non esiste nella posizione indicata\n",argv[1]);
        exit(-3);
    }
    /*                                                  <------->
       [...\n...\n....\n....\n\0] --> [.............\0] --> [.......]\n
    */
    while(fgets(riga,sizeof(riga),file)!=NULL)
    {
        int scritti;
        if (riga[strlen(riga)-1] == '\n')
            riga[strlen(riga)-1]  = ' ';
        scritti = sprintf(pCh, "%s", riga);
        pCh += scritti;
    }
    pCh = articolo_base;  // pCh = &(articolo_base[0]);
    lunghezza_articolo = strlen(articolo_base);

    while (pCh < (articolo_base + lunghezza_articolo)) {
      int j = dimensione;
      x++;
      if (strlen(pCh) <= j) {
          printf("%s\n", pCh);
          break;
      };
      if (pCh[0] == ' ') pCh++;
      if (pCh[j] == ' ') {
          printf("%.*s\n", j, pCh);
          pCh += j;
      } else {
          while (pCh[j-1] != ' ') j--;
          printf("%.*s\n", j, pCh);
          pCh += j;
      }
    }
    fclose(file);
    return x;
}
```