SO Cheatsheet

Bozza file bash (presa dalla prima simulazione d'esame)

```
#!/usr/bin/env bash

# bozza soluzione file "bash"

# se sono passati due argomenti e il primo è un file prova
# a rimuoverlo ignorando eventuali errori (*)
if [[ $# -eq 2 ]] && [[ -f "$1" ]]; then rm "$1" 1>/dev/null 2>&1; fi

# richiama "make" passando come argomento il nome desiderato
# per l'eseguibile ignorando eventuali avvisi/errori (*)
# e poi richiama l'eseguibile passando tutti gli argomenti
# per poi mostrare il contenuto del file di "log"
make NAME=program 1>/dev/null 2>&1 && ./program $@ && cat "$1"
# (*) in una variante che gestisca tutti i casi sarebbe opportuno
#     gestire i vari casi d'errore che possono verificarsi dentro
#     questo script.
```

Lettura di un tasto da tastiera senza echo

```
#include <stdio.h>
#include <unistd.h>
#include <termios.h>
typedef enum {
   KP_ECHO_OFF,
   KP_ECHO_ON,
} kp_echo_t;
int keypress(const kp_echo_t echo) {
   struct termios savedState, newState;
   unsigned char echo_bit; // flag
   int c;
   if (-1 == tcgetattr(STDIN_FILENO, &savedState)) { return EOF; }; // error
   newState = savedState;
   if (KP_ECHO_OFF == echo) { echo_bit = ECHO; } else { echo_bit = 0; };
   /* canonical input + set echo with minimal input as 1. */
   newState.c_lflag &= ~(echo_bit | ICANON);
   newState.c_cc[VMIN] = 1;
   if (-1 == tcsetattr(STDIN_FILENO, TCSANOW, &newState)) { return EOF; }; // error
   c = getchar(); /* block until key press */
   if (-1 == tcsetattr(STDIN_FILENO, TCSANOW, &savedState)) { return EOF; }; // error
   return c;
}
```

```
int main() {
    char c;
    while (1) {
        c = keypress(KP_ECHO_OFF); // read single keypress without echoing
        if (c=='+') {
            printf("PLUS\n");
        };
        if (c=='-') {
            printf("MINUS\n");
        };
        if (c=='\n') {
            printf("ENTER\n"); break;
        };
    };
    return 0;
}
```

Gestione segnali avanzata

```
struct sigaction sa; // imposta variabile
  sigemptyset(&sa.sa_mask); // imposta maschera vuota di attributi
  sa.sa_flags |= SA_SIGINFO; sa.sa_flags |= SA_RESTART;
// permette di leggere info e di riavviare eventuali funzioni particolari (*)
  sa.sa_sigaction = handlerManager; // imposta l'handler
  sigaction(SIGUSR1, &sa, NULL); // attiva l'handler
```

Esempio di handler

```
void sigHandler(int signo, siginfo_t *info, void *empty) {
    char spid[PID_LEN];
    int sender = info->si_pid; //recupero il "pid" del processo che ha inviato il segnale
   ...
```

Gestione segnali base

```
signal(sig_id, sig_handler)
```

Uso di booleani

```
typedef enum { false = 0, true = 1 } bool;
```

oppure includere <stdbool.h>

Makefile esempio

```
# Creare un makefile con:
# - una regola "help" di default che mostri una nota informativa,
# - una regola "backup" che crei un backup di una cartella appendendo “.bak” al nome,
# - una regola "restore" che ripristini il contenuto originale.
# Per definire la cartella sorgente passarne il nome come variabile,
# ad esempio:
#   make -f mf-backup FOLDER=...
#   (la variabile FOLDER è disponibile dentro il makefile)
#
# In queesta versione il "backup" di una cartella avviene
# copiando l'intera cartella in un'altra con ".bak" aggiunto al nome,
# mentre il "restore" avviene rinominando la versione ".bak" e quindi
# eliminando il backup stesso, sempre che non esista già l'originale.
# Si può espandere l'esempio per aggiungere maggiori controlli e opzioni.

# .SILENT dichiara le regole di cui NON fare l'echo
# evitando di usare "@" prima di ogni comando
.SILENT: help backup restore

# .PHONY dichiara le pseudo-regole (con target non file
# o comunque con target da non verificare)
.PHONY: help backup restore

# per comodità si definiscono alcune variabili
NAME=mf-backup
APP=make -f $(NAME)

# regola "help" di default
help:
	echo
	echo backup/restore folder
	echo "  $(APP) backup  FOLDER=..."  # backup folder (create .bak)
	echo "  $(APP) restore FOLDER=..." # restore folder (retrieve from .bak)
	echo

# regola "backup":
# verifica esista la cartella sorgente e crea una copia con ".bak"
backup:
	echo "Backup folder '$(FOLDER)'"
	[ -d "$(FOLDER)" ] && cp -rp "$(FOLDER)" "$(FOLDER).bak" || echo "?Error"

# regola "restore":
# verifica esista la versione ".bak" e non ci sia l'orginiale e ripristina il contenuto
restore:
```

```
	echo "Restore folder '$(FOLDER)'"
	[ -d "$(FOLDER).bak" ] && [ ! -d "$(FOLDER)" ] && mv "$(FOLDER).bak"
"$(FOLDER)" || echo "?Error"
```

Librerie comode da includere

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <sys/wait.h>
#include <fcntl.h>
```

Blocco di un segnale

```
void block_signal(int sig)
{
   sigset_t s;
   sigaddset(&s, sig);
   sigprocmask(SIG_BLOCK, &s, NULL);
}
```

Funzione switch per gestire gli errori
NOTA BENE: cambia i valori degli errori in modo opportuno!!!

```
void quit(int errNo){
   switch (errNo)
   {
   case 0:
      break;
   case 3:
      fprintf(stderr, "Incorrect input. Usage: ./NAME <target> <n>.\n");
      break;

   case 4:
      fprintf(stderr, "File already exists or directory given is invalid.\n");
      break;
   case 5:
      fprintf(stderr, "Incorrect input. <n> must be between 1 and 10.\n");
      break;
   case 6:
      fprintf(stderr, "Wfork failure.\n");
      break;
   case 7:
      fprintf(stderr, "Pipe reading error.\n");
      break;
   case 8:
```

```
        fprintf(stderr, "sprintf error.\n");
        break;
    case 9:
        fprintf(stderr, "Pipe writing error.\n");
        break;
    default:
        fprintf(stderr, "Unknown error.\n");
        break;
    }
    exit(errNo);
}
```

Funzione wfork per facilitare la gestione dei fork

```
int wfork(){
    int f=fork();
    if(f<0) quit(6);
    return f;
}
```

Funzione che controlla se il file esiste

```
void testIfFileExists(char * filePath){
    FILE * fp1 = fopen(filePath, "r");
    if(fp1 != NULL){
        fclose(fp1);
        quit(4);
    }

}
```