Sistemi Operativi, Secondo Modulo
A.A. 2018/2019
Testo del Primo Homework

Emiliano Casalicchio, Igor Melatti

Come si consegna

Il presente documento descrive le specifiche per l'homework 1. Esso consiste in 3 esercizi, per risolvere i quali occorre scrivere 3 files che si dovranno chiamare `1.sh` (soluzione del primo esercizio), `2.sh` (soluzione del secondo esercizio) e `3.awk` (soluzione del terzo esercizio). Per consegnare la soluzione, seguire i seguenti passi:

1. creare una directory chiamata `so2.2018.2019.1.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare `1.sh`, `2.sh` e `3.awk` in `so2.2018.2019.1.matricola`
3. creare il file da sottomettere con il seguente comando: `tar cfz so2.2018.2019.1.matricola.tgz {1..3}.*`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=61` e uploadare il file `so2.2018.2019.1.matricola.tgz` ottenuto al passo precedente.

Come si auto-valuta

Per poter autovalutare il proprio homework, si hanno 2 possibilità:

- usare la macchina virtuale Debian-9 del laboratorio "P. Ercoli";
- installare VirtualBox (`https://www.virtualbox.org/`), e importare il file OVA scaricabile dall'indirizzo `https://drive.google.com/open?id=1LQORjuidpGGt9UMrRupoY73w_qdAoVCp`; maggiori informazioni sono disponibili all'indirizzo `http://twiki.di.uniroma1.it/twiki/view/SO/SO1213AL/SistemiOperativi12CFUModulo220182019#software`. Si tratta di una macchina virtuale quasi uguale a quella del laboratorio. Si consiglia di configurare la macchina virtuale con NAT per la connessione ad Internet, e di settare una "Shared Folder" (cartella condivisa)

per poter facilmente scambiare files tra sistema operativo ospitante e Debian. Ovvero: tramite l'interfaccia di VirtualBox, si sceglie una cartella x sul sistema operativo ospitante, gli si assegna (sempre dall'interfaccia) un nome y, e dal prossimo riavvio di VirtualBox sarà possibile accedere alla cartella x del sistema operativo ospitante tramite la cartella `/media/sf_`y di Debian.

All'interno delle suddette macchine virtuali, scaricare il pacchetto per l'autovalutazione (*grader*) dall'URL `151.100.17.205/download_from_here/so2.grader.1.20182019.tgz` e copiarlo in una directory con permessi di scrittura per l'utente attuale. All'interno di tale directory, dare il seguente comando:

```
tar xfzp so2.grader.1.20182019.tgz && cd grader.1
```

È ora necessario copiare il file `so2.2018.2019.1.matricola.tgz` descritto sopra dentro alla directory attuale (ovvero, `grader.1`). Dopodiché, è sufficiente lanciare `grader.1.sh` per avere il risultato: senza argomenti, valuterà tutti e 3 gli esercizi, mentre con un argomento pari ad i valuterà solo l'esercizio i (in quest'ultimo caso, è sufficiente che il file `so2.2018.2019.1.matricola.tgz` contenga solo l'esercizio i).

Dopo un'esecuzione del grader, per ogni esercizio $i \in \{1, 2, 3\}$, c'è un'apposita directory `input_output.`i contenente le esecuzioni di test. In particolare, all'interno di ciascuna di tali directory:

- sono presenti dei file `inp_out.`j`.sh` ($j \in \{1, \ldots, 6\}$ per gli esercizi 1 e 2, $j \in \{1, \ldots, 8\}$ per l'esercizio 3) che eseguono la soluzione proposta con degli input variabili;
- lo standard output (rispettivamente, error) di tali script è rediretto nel file `inp_out.`j`.sh.out` (rispettivamente, `inp_out.`j`.sh.err`);
- l'input usato da `inp_out.`j`.sh` è nella directory `inp.`j;
- l'output creato dalla soluzione proposta quando lanciata da `inp_out.`j`.sh` è nella directory `out.`j;
- nella directory `check` è presente l'output corretto, con il quale viene confrontato quello prodotto dalla soluzione proposta.

Nota bene: per evitare soluzioni "furbe", le soluzioni corrette nella directory `check` sono riordinate a random dal grader stesso. Pertanto, ad esempio, `out.1` potrebbe dover essere confrontato con `check/out.5`. L'output del grader mostra di volta in volta quali directory vanno confrontate.

Esercizio 1

Scrivere uno script `1.sh` con la seguente sinossi:

`1.sh [opzioni] directory`

dove le opzioni sono le seguenti (si consiglia l'uso del comando bash `getopts`, vedere `http://wiki.bash-hackers.org/howto/getopts_tutorial`):

- -e
- -b dir (default: vuoto; nel seguito, sia b il valore dato a tale opzione)

L'invocazione dello script è da considerarsi sbagliata nei seguenti casi:

- viene passata un'opzione non esistente (ovvero, non compresa in quelle elencate sopra);
- viene passata un'opzione che necessita un argomento, ma senza passare l'argomento;
- vengono passate entrambe le opzioni `-e` e `-b`;
- non viene passato l'argomento obbligatorio.

Se si verifica uno dei casi di errore appena elencati, l'output dovrà consistere nella sola riga, su standard error, `Uso:` s `[opzioni] directory`, con s nome dello script, e lo script dovrà terminare con exit status 10.

Nel seguito, sia d il valore dato all'argomento dello script. Se d non esiste, o è un file regolare e non una directory, o non ha entrambi i permessi di lettura ed esecuzione, l'output dovrà essere semplicemente la scritta `L'argomento` d `non e' valido in quanto` x su standard error, con exit status 100, dove x è la spiegazione dell'errore stesso. L'argomento opzionale b, quando dato, deve o essere una directory esistente, nel qual caso deve avere i diritti corrispondenti all'ottale 7 (sempre per l'utente attuale, gli altri utenti non devono avere alcun diritto). Il fatto che non esista, invece, non costituisce errore: occorre in tal caso crearla con permessi `rwx------`. In caso si manifesti uno degli errori sopra descritti, il codice di uscita dev'essere 200, e l'output dovrà essere semplicemente la scritta `L'argomento` b `non e' valido in quanto` x su standard error.

Lo script deve cercare tutti i file che si trovano nel sottoalbero di directory radicato in d, limitandosi solamente ai file i nomi dei quali contengono una data del tipo $AAAAMMGGHHMM$ (anno, mese, giorno, ora, minuti), e terminano con l'estensione `jpg` oppure `txt` (minuscola o maiuscola). La data è sempre preceduta e seguita dal carattere _, che non occorre in nessun altro punto del nome del file. Per ogni gruppo di file F che contiene nel nome la stessa data, ora e minuti, è necessario individuare 3 sottoinsiemi $F', F'', F''' \subset F$ tali che valgano le seguenti condizioni:

- per ogni $f \in F'$, f è un link simbolico ad un $g \in F$;

- per ogni $f \in F''$, f è un hard link ad un $g \in F \setminus F'$; inoltre, per ogni $h \in F \setminus F'$ tale che f è un hard link ad h, il path di f è lessicograficamente maggiore di quello di h;
- per ogni $f \in F'''$, esiste un $g \in F \setminus F' \setminus F''$ per il quale f e g hanno lo stesso contenuto; inoltre, esiste un $h \in F \setminus F' \setminus F''$ tale che f ed h hanno lo stesso contenuto, ed il path di h è lessicograficamente maggiore di quello di f. Altrimenti detto, se F'''' è l'insieme dei file in $F \setminus F' \setminus F''$ aventi lo stesso contenuto, allora F''' è costituito da tutti i file di F'''' tranne uno, ovvero quello con path lessicograficamente maggiore.

Tutti i file in un qualche F', F'', F''' vanno eliminati. Prima di ciò, occorre scrivere sullo standard output la lista dei file che stanno per essere eliminati (tutti sulla stessa riga, separati dal carattere `|` ed ordinati lessicograficamente sul path), con il loro path relativo rispetto a d.

Se b non è vuoto, i file, anziché cancellati, vanno spostati nella directory b, mantenendo il loro path relativo rispetto a d.

Se, cancellando o spostando files, si dovessero creare dei link simbolici con destinazione non esistente, occorre eliminare tali link.

Se è stata data l'opzione `-e`, occorre solo scrivere su standard output la lista dei file da eliminare (tutti sulla stessa riga, separati dal carattere `|` ed ordinati lessicograficamente), senza né cancellare né spostare nulla.

Nota bene: per "ordine lessicografico" si intende quello usato da `awk` e `sort` per ordinare stringhe. In caso si usi `sort`, accertarsi di usare l'ordinamento "tradizionale", che si basa sui "native bytes values" (vedere il `man`).

Attenzione: non è permesso usare Python, Java, Perl o GCC. Lo script non deve scrivere nulla sullo standard error, a meno che non si tratti di un errore nelle opzioni da riga di comando come descritto sopra. Inoltre, non deve scrivere nulla sullo standard output, tranne che nei casi indicati sopra. Per ogni test definito nella valutazione, lo script dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.1`, dare il comando `tar xfzp all.tgz input_output.1 && cd input_output.1`. Ci sono 6 esempi di come lo script `1.sh` può essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 6\}$). Per ciascuno di questi script, la directory di input è `inp.`i, e la directory con l'output atteso è `check/out.`i; lo standard output atteso sarà nel file `check/inp_out.`i`.sh.out`, mentre lo standard error atteso sarà nel file `check/inp_out.`i`.sh.err`.

Esercizio 2

Scrivere uno script bash con i seguenti argomenti (nell'ordine dato):

1. numero di bytes b
2. walltime w in minuti;
3. intervallo di campionamento c;
4. lista di comandi C da lanciare con rispettivi argomenti, terminati da un doppio punto e virgola (`;;`); l'ultimo comando va terminato con un triplo punto e virgola (`;;;`);
5. lista di nomi di file.

Se uno dei suddetti input manca (o ve ne sono in eccesso), l'output dovrà essere semplicemente la scritta `Usage:` s `bytes walltime sampling commands files` su standard error (dove s è il nome dello script), e lo script dovrà terminare con exit status 15.

Lo script dovrà lanciare in background e monitorare i comandi in C, ridirigendo sia lo standard output che lo standard error. A tal proposito, la lista di nomi di file data come ultimo argomento dovrà essere del tipo $f_1, f_2, \ldots, f_n, g_1, \ldots, g_n$, dove n è il numero di comandi in C. Sia $C = C_1, \ldots, C_n$; allora lo standard output del comando C_i va rediretto in f_i e lo standard error in g_i. Una mancata concordanza tra numero di comandi e numero di file dovrà portare lo script a terminare con exit status 30, visualizzando su standard error `Usage:` s `bytes walltime sampling commands files` (dove s è il nome dello script). Qualora uno dei comandi in C non esista, non va lanciato. Una volta lanciati tutti i comandi, occorre scrivere sul file descriptor 3 i PID dei processi lanciati (tutti sulla prima riga, separati da uno spazio), *prima* di iniziare il monitoraggio descritto qui sotto. Se uno dei processi monitorati supera b bytes come dimensione totale della sua immagine, oppure ha un elapsed time che supera w minuti, va killato simulando con un segnale la pressione di CTRL+C. Il controllo su tali condizioni va effettuato ogni c secondi. Il controllo sulla dimensione dell'immagine del processo va effettuato solo se $b > 0$, e il controllo sull'elapsed time solo se $w > 0$. Lo script, se non riscontra errori, deve rimanere in esecuzione finché non trova un file regolare `done.txt` sulla current working directory, oppure fino a quando non terminano tutti i processi in C. Nel primo caso, deve scrivere `File done.txt trovato` sullo standard output, con exit status 0; nel secondo, deve scrivere `Tutti i processi sono terminati` sullo standard output, con exit status 1.

Attenzione: non è permesso usare Python, Java, Perl o GCC. Lo script non deve scrivere nulla sullo standard error, a meno che non si tratti di un errore nelle opzioni da riga di comando come descritto sopra. Non deve mai scrivere nulla sullo standard output, tranne che nei casi descritti sopra. Per ogni test definito nella valutazione, lo script dovrà ritornare la soluzione dopo al più 10 minuti, e comunque lanciare i comandi passatigli entro al più 3 secondi.

Esempi

Da dentro la directory `grader.1`, dare il comando `tar xfzp all.tgz input_output.2 && cd input_output.2`. Ci sono 6 esempi di come lo script `2.sh` può essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 6\}$). Per ciascuno di questi script, la directory `inp.`i contiene uno script `main.sh` e degli altri file necessari per lanciare `2.sh` e controllare che funzioni correttamente. La directory `check/out.`i contiene i file con l'output atteso; lo standard output atteso sarà nel file `check/inp_out.`i`.sh.out`, mentre lo standard error atteso sarà nel file `check/inp_out.`i`.sh.err`.

Esercizio 3

LaTeXè un linguaggio di markup per la preparazione di testi, prediletto dagli autori di testi scientifici (questo documento è stato scritto con LaTeX). Per gli scopi di questo esercizio, basti sapere che occorre scrivere un file `F.tex` seguendo una particolare sintassi, invocare il comando `pdflatex F.tex` e, se non ci sono errori, verrà creato il file `F.pdf`. Contestualmente, viene anche creato un file `F.log`, che contiene informazioni utili riguardo alla fase di generazione di `F.pdf`.

La cosa bella del LaTeXè che è possibile fare in modo che F includa al suo interno altri file, che a loro volta contengono parti del documento da creare (per esempio, si può fare un file per ogni diverso capitolo del documento), e ovviamente delle immagini. Tuttavia, nel caso di documenti particolarmente complessi, è possibile che ci siano file, inizialmente inclusi nel documento, che poi ne vengono esclusi, senza però cancellarli.

Spesso viene chiesto, per la pubblicazione di documenti LaTeX, di fornire tutti i file sorgenti, compresi quindi quelli inclusi a partire da F e le immagini. Inoltre, è spesso importante rimuovere i commenti, che in LaTeXcominciano con il carattere `%` e finiscono al termine della riga. Per scrivere lo stesso carattere `%`, è necessario scrivere `\%`, che quindi ovviamente non va considerato l'inizio di un commento.

Si richiede quindi di scrivere uno script `gawk`, da chiamare `3.awk`, che prenda in input n file di testo, che nel seguito indicheremo con $I_1, \ldots, I_n$. Il file I_1 contiene delle configurazioni per awk, formattate come segue:

```
strip_comments=x
only_figs=y
also_figs=z
```

L'ordine delle righe di I_1 può essere qualsiasi, e $x, y, z \in \{0, 1\}$ sono valori booleani. Nel caso una qualche riga manchi, il corrispondente valore è da intendersi 0.

Il file I_2, invece, è un file di log `F.log` di una compilazione LaTeX. Per i nostri fini, basti sapere quanto segue.

- Ogni file incluso a partire da F è presente in `F.log` con la seguente sintassi: `(nomefile` (potrebbero poi seguire altre stringhe non di interesse, prima della parentesi chiusa).
- Il `nomefile` di cui sopra potrebbe essere spezzato su più righe, se troppo lungo. Per i nostri scopi, una riga contenente un nome di un file non è completa se non finisce in alcuno dei seguenti modi: `.tex`, `.cls`, `.sty`, `.bbl` o `.aux`, seguiti o no da una parentesi tonda chiusa.
- Occorre considerare solo i `nomefile` il cui path ha come primo carattere un punto.

- Ogni immagine inclusa a partire da F è presente in F`.log` con la seguente sintassi: `File: nomeimmagine`.
- Come sopra, occorre considerare solo i `nomeimmagine` il cui path comincia con un punto, ma in questo caso il punto deve essere seguito da uno slash `./`.
- Il `nomeimmagine` di cui sopra potrebbe essere spezzato su più righe, se troppo lungo. Per i nostri scopi, una riga contenente un nome di un file non è completa se non finisce in alcuno dei seguenti modi: `.png`, `.jpg`, o `.pdf`, seguiti o no da una parentesi tonda chiusa.

Lo script `3.awk` dovrà quindi scrivere sia su standard output che su standard error, come prima cosa, la sequenza dei file passatigli come argomento, tutti su una riga separati da spazi e preceduti dalla stringa `Eseguito con argomenti`. Dopodiché, dovrà stampare, solo su standard output, la lista dei file con estensione `.tex` (uno per ogni riga, nell'ordine con il quale compaiono in I_2) effettivamente inclusi a partire da F, assumendo appunto che I_2 sia il file di log F`.log` ottenuto compilando F`.tex`. Se `also_figs` è vero, deve riportare anche le figure; se `only_figs` è vero, deve riportare *solo* le figure. Se `strip_comments` è vero, allora occorre considerare i file $I_3, \ldots, I_n$, e *modificarli* in modo tale da eliminare i commenti. Ovvero: per ogni riga che contenga, da un certo punto in poi, il carattere `%` *non* preceduto da \, occorre cancellare tutto quello che va dal primo carattere `%` in poi. Se il carattere `%` è all'inizio di una riga, oppure è preceduto da soli spazi, occorre eliminare l'intera riga.

Lo script non deve scrivere nulla sullo standard error, tranne che nei seguenti casi:

- nella prima riga, per riportare gli argomenti come descritto sopra;
- non vengono dati almeno 2 file di input; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: dare almeno 2 file di input`, e terminare senza effettuare altre computazioni con exit status 0;
- `only_figs` è vero, ma `also_figs` non lo è; in tal caso, lo script deve scrivere su standard error il messaggio `Errore di configurazione: only_figs=1 e also_figs=0`, e terminare senza effettuare altre computazioni con exit status 0;
- `strip_comments` è vero, ma uno dei file $I_3, \ldots, I_n$, sia esso I_j, non è tra i file effettivamente inclusi da F; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: il file` I_j `non risulta incluso`, e proseguire senza modificare I_j. Al termine di tutti i file, l'exit status deve essere pari al numero di file non inclusi trovati.

Attenzione: per ogni test definito nella valutazione, lo script dovrà ritornare la soluzione dopo al più 10 minuti.

Esempi

Da dentro la directory `grader.1`, dare il comando `tar xfzp all.tgz input_output.3 && cd input_output.3`. Ci sono 8 esempi di come lo script `3.awk` può essere lanciato, salvati in file con nomi `inp_out.`i`.sh` (con $i \in \{1, \ldots, 8\}$). Per ciascuno di questi script, la directory `inp.`i contiene i file di input. La directory `check/out.`i contiene i file con l'output atteso; lo standard output atteso sarà nel file `check/inp_out.`i`.sh.out`, mentre lo standard error atteso sarà nel file `check/inp_out.`i`.sh.err`.