Sistemi Operativi, Secondo Modulo
A.A. 2019/2020
Testo del Primo Homework

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 `so2.2019.2020.1.matricola`, dove al posto di `matricola` occorre sostituire il proprio numero di matricola;
2. copiare `1.sh`, `2.sh` e `3.awk` in `so2.2019.2020.1.matricola`
3. da dentro la directory `so2.2019.2020.1.matricola`, creare il file da sottomettere con il seguente comando: `tar cfz so2.2019.2020.1.matricola.tgz {1..3}.*`
4. andare alla pagina di sottomissione dell'homework `151.100.17.205/upload/index.php?id_appello=88` e uploadare il file `so2.2019.2020.1.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" (stanti le ultime limitazioni, questa possibilità è purtroppo da scartare);
- 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/SistemiOperativi12CFUModulo220192020#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.1.20192020.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.1.20192020.tgz && cd grader.1
```

È ora necessario copiare il file `so2.2019.2020.1.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 `so2.2019.2020.1.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 tutti gli esercizi 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] stringa file1... filen`

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

- -r
- -e regex (default: vuoto; nel seguito, sia e 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;
- viene passata l'opzione `-r` senza l'opzione `-e`;
- non vengono passati almeno 2 argomenti.

Se si verifica uno dei casi di errore appena elencati, l'output dovrà consistere nella sola riga, su standard error, `Uso:` s `[opzioni] stringa file1... filen`, con s nome dello script, e lo script dovrà terminare con exit status 10.

Nel seguito, siano $s, f_1, \ldots, f_n$ i valori dati agli argomenti obbligatori dello script. Occorre controllare le seguenti condizione di errore.

- Per ogni f_i che non esiste occorre scrivere `L'argomento` f_i `non esiste` su una riga separata sul file descriptor 3; la computazione deve poi continuare ignorando f_i.
- Se l'opzione `-r` non è stata data, per ogni f_i che è una directory, occorre scrivere `L'argomento` f_i `e' una directory` su una riga separata sul file descriptor 4; la computazione deve poi continuare ignorando f_i.
- Se l'opzione `-r` è stata data, per ogni f_i che è una directory ma non ha, per il proprietario, i permessi di lettura, scrittura ed esecuzione, con anche il setgid bit per il gruppo, occorre scrivere `I permessi` x `dell'argomento` f_i `non sono quelli richiesti` su una riga separata sul file descriptor 5 (x deve essere il numero ottale corrispondente a permessi e attributi speciali); la computazione deve poi continuare ignorando f_i.

Lo script deve cercare in tutti i file f_i (esclusi quelli da ignorare come descritto sopra) il contenuto *esadecimale* s, considerando per l'appunto il contenuto bit a bit dei file stessi (convertito in esadecimale). Se la dimensione S del file non è un multiplo di 4 bytes, riempire gli ultimi S mod 4 bytes con zeri. Se è stata data l'opzione `-r`, per ogni f_i che sia una directory occorre fare la

ricerca su tutti i file regolari che si trovano nell'albero della directory radicata in f_i, e il cui nome fa pattern matching con la ERE e (sintassi di `grep`). Se nell'output compaiono 2 o più file che sono l'uno il link simbolico dell'altro, considerare solo i link simbolici e non i file cui tali link fanno riferimento. Se nell'output compaiono 2 o più file che sono hard link allo stesso inode, considerare solo il file con il nome più corto (solo il nome del file, non tutto il path); se la lunghezza minima risulta n e più di un file ha un nome di lunghezza n, considerare tutti tali file.

Per ogni match di contenuto trovato in un file f_i, lo script dovrà scrivere su standard output su una riga separata la stringa f_i`:`n, dove n è l'offset del byte da cui comincia il match stesso. L'intero output (anche quello sugli altri file descriptor) va ordinato lessicograficamente (riga per riga) in modo decrescente; eventuali righe ripetute vanno riportate una sola volta. L'exit status dello script deve essere il numero di file o directory, tra gli argomenti dello script, che sono stati ignorati secondo le regole di cui sopra. Inoltre, l'exit status va anche scritto in ottale su standard error.

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 uno dei casi descritti esplicitamente sopra. Analogamente, non deve scrivere nulla sullo standard output, tranne che nei casi indicati esplicitamente 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. intervallo di campionamento c;
2. lista di comandi C da lanciare con rispettivi argomenti, terminati da una doppia virgola (`,,`); l'ultimo comando va terminato con una tripla virgola (`,,,`);
3. lista di nomi di file.

Se uno dei suddetti input manca, l'output dovrà essere semplicemente la scritta `Uso:` s `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 `Uso:` s `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 dal carattere _), *prima* di proseguire con la computazione descritta qui sotto. Lo script, se non riscontra errori, deve rimanere in esecuzione finché non trova un file regolare `done.txt` sulla current working directory. Il monitoraggio di questa condizione deve avvenire ogni c secondi. A quel punto, deve scrivere sul file descriptor 3 `File done.txt trovato`, seguito dalla scrittura su standard output della foresta di processi di C, per poi terminare con exit status 0. La foresta di processi va scritta come una sequenza di righe, ognuna delle quali contiene due PID separati da uno spazio: il primo di un processo padre, il secondo di un processo figlio. Le radici della foresta devono corrispondere a tutti e soli i processi di C. Le righe vanno ordinate (numericamente) prima sul primo PID, e poi sul secondo.

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 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`. Ovviamente, l'output della directory `check` difficilmente coinciderà con l'output corretto (i PID cambiano...), ma può essere usata come riferimento per la formattazione dell'output stesso.

Esercizio 3

Le moderne applicazioni per gestire fogli di calcolo (ad es. Microsoft Excel o Libreoffice Calc) permettono il salvataggio dei file in formato CSV. Tale formato è testuale, e semplicemente usa un opportuno carattere separatore (tipicamente la virgola, da cui il nome Comma Separated Values) per scrivere il contenuto di ogni cella (ovviamente, questo funziona solo se si è sicuri che non ci siano celle che contengono quel carattere). Da notare che eventuali formule vengono memorizzate direttamente con il loro risultato; altri elementi (figure etc.) vengono ignorate.

Nel seguito, supponiamo che ogni file CSV contenga n tabelle di un database; le prime 2 righe sono di intestazione e dicono rispettivamente come si chiama la tabella e come si chiamano i vari campi (ovvero, gli attributi della tabella). Due tabelle nello stesso file CSV sono separate da una o più righe vuote (che, nel file CSV, sono rappresentate da una riga con soli $m-1$ caratteri di separazione, se m è il numero di attributi). Da notare che un file CSV ha sempre lo stesso numero di campi, pari quindi al massimo numero di attributi di una tabella al suo interno.

Si richiede quindi di scrivere uno script `gawk`, da chiamare `3.awk`, che prenda in input un file di configurazione ed n file CSV; nel seguito, indicheremo tali file con $I, F_1, \ldots, F_n$ (da considerare nell'ordine con cui vengono dati in input). Il file I è formattato come segue:

`separator_`j=s_j
`num_pivot_`t=k_t
`string_pivot_`t=p_t
`tab_1_1`=$\mathcal{T}_{11}$
`tab_1_2`=$\mathcal{T}_{12}$
`tab_2_1`=$\mathcal{T}_{21}$
`tab_2_2`=$\mathcal{T}_{22}$
...
`tab_`m`_1`=$\mathcal{T}_{m1}$
`tab_`m`_2`=$\mathcal{T}_{m2}$

L'ordine delle righe di I può essere qualsiasi. Gli j, k_t sono numeri interi, mentre s_j e p_t sono stringhe. Se c'è un t t.c. t non è il nome di una tabella nell'input, oppure un $j > n \vee j \leq 0$ occorre ignorare la rispettiva riga. Se per un qualche t valido sono specificati sia k_t che p_t, considerare definito solo quello che appare per ultimo. Se per un qualche j valido c'è un s_j specificato più volte, considerare solo l'ultima occorrenza; fare lo stesso per i k_i e p_i. Se per un qualche j, s_j non è specificato, occorre assegnare ad s_j il carattere virgola `,`. Se per un qualche t, né k_t né p_t sono specificati, considerare definito k_t con valore 1. Se una riga non è formattata nei modi indicati sopra, occorre ignorarla. Nel seguito, supporremo p_t sempre definito per ogni t: nel caso in cui per un t sia

definito k_t anziché p_t, occorre assumere p_t definito come l'attributo k_t-esimo della tabella t-esima.

Lo script `3.awk` dovrà quindi scrivere sia su standard output che su standard error, come prima cosa, la sequenza dei nome dei file passatigli come argomento, tutti su una riga separati da spazi e preceduti dalla stringa `Eseguito con argomenti`. Dopodiché, dovrà stampare, aggiungendolo in coda all'ultimo file F_n, il risultato delle m formule SQL `SELECT * from` $\mathcal{T}_{i1}$ `INNER JOIN` $\mathcal{T}_{i2}$ `ON` $\mathcal{T}_{i1}.p_{\mathcal{T}_{i1}}$ `=` $\mathcal{T}_{i2}.p_{\mathcal{T}_{i1}}$, per $1 \leq i \leq m$. Note:

- Ogni risultato di join deve essere preceduto da 2 righe: una vuota, e la successiva composta di 4 campi $\mathcal{T}_{i1}, k_{\mathcal{T}_{i1}}, \mathcal{T}_{i2}, k_{\mathcal{T}_{i2}}$. Queste due righe dovranno essere sempre presenti, anche se il risultato del join è vuoto.
- L'aggiunta dei risultati alla fine di F_n potrebbe richiedere di modificare l'intero file F_n, in quanto il risultato di questa operazione potrebbe contenere più attributi delle tabelle presenti in F_n.
- $\mathcal{T}_{i1}, \mathcal{T}_{i2}$ sono i nomi delle tabelle coinvolte, e $p_{\mathcal{T}_{i1}}, p_{\mathcal{T}_{i2}}$ sono i nomi degli attributi-pivot per l'i-esimo join tra tali tabelle, come definiti nel file di configurazione I.
- Nel risultato, prima devono essere presenti gli attributi di $\mathcal{T}_{i1}$ e poi quelli di $\mathcal{T}_{i2}$, ciascuno ordinato alfabeticamente.

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 10;
- non viene specificata, nel file di configurazione, neanche una tabella con prefisso `tab_`; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: non e' stata precisata neanche una tabella per il join`, e terminare senza effettuare altre computazioni con exit status 20;
- per qualche i non vengono specificati, nel file di configurazione, entrambi i parametri `tab_`i`_1` e `tab_`i`_2`; oppure, per qualche $i > 1$, esiste almeno un parametro `tab_`i`_1` o `tab_`i`_2` ma non i parametri `tab_`$(i-1)$`_1` e `tab_`$(i-1)$`_2`; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: non sono precisate le tabelle del join all'indice` i (con i indice minimo dell'errore), e terminare senza effettuare altre computazioni con exit status 30;
- un $x \in \{\mathcal{T}_{ij} \mid 1 \leq i \leq m \wedge j \in \{1,2\}\}$ non esiste nell'input; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: la`

`tabella` x `non e' presente nell'input`, e terminare senza effettuare altre computazioni con exit status 35;

- un $x \in \{k_1, \ldots, k_q\}$ non è nel range $[1, r]$, con r numero di attributi della corrispettiva tabella y; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: la posizione dell'attributo` x `non e' corretta per la tabella` y, e terminare senza effettuare altre computazioni con exit status 40;

- un $x \in \{p_1, \ldots, p_\ell\}$ non è tra gli attributi della corrispettiva tabella y; in tal caso, lo script deve scrivere su standard error il messaggio `Errore: l'attributo` x `non e' corretto per la tabella` y, e terminare senza effettuare altre computazioni con exit status 50.

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`.