Esercitazione Perl

Esercizio 1.

Realizzare uno script perl `esa.pl` che elabora dei numeri e restituisce su standard output (per ciascun numero) una stringa che specifica se il numero inserito è esadecimale o meno.
Lo script:

1. processa iterativamente i numeri che vengono forniti
   - come argomenti da linea di comando, **oppure**
   - da standard input (se non sono esplicitamente presenti degli argomenti da linea di comando)
2. restituisce su standard output una stringa che indica se il numero inserito è un numero esadecimale o meno.

Suggerimenti: un numero esadecimale è costituito dalle cifre da 0 a 9 più le lettere (maiuscole o minuscole) dalla A alla F.

Per esempio se da linea di comando si scrive:

```
$./esa.pl BABBA02BA T04
```

lo script stampa in output:

```
Il numero BABBA02BA è esadecimale
Il numero T04 non è esadecimale
```

Se invece da linea di comando si scrive:

```
$./esa.pl
```

lo script chiede di inserire il numero da valutare:

```
Inserire il numero:
```

se viene inserito il numero `BABBA02BA` lo script stampa in output:

```
Numero esadecimale!
```

se invece viene inserito `T04` lo script stampa in output:

```
Il numero non è esadecimale.
```

Esercizio 2.

Si vuole realizzare uno script perl per implementare un comando `pingscan <ipiniziale> <ipfinale>`. Lo script `pingscan` riceve due argomenti (da linea di comando), `<ipiniziale>` e `<ipfinale>`, che rappresentano un *intervallo* di indirizzi IP. Lo script deve invocare il comando *ping* su ciascuno degli indirrizzi IP compresi nell'intervallo [`ipiniziale, ipfinale`] e stampare a video una stringa del tipo: `IP X.X.X.X: OK`, se l'indirizzo `X.X.X.X` è raggiungibile, oppure `IP X.X.X.X: FAIL`, se non lo è.

Nota: per semplicità si assume che l'intervallo degli indirizzi IP sia così definito [`X.X.X.A`, `X.X.X.B`], ovvero, dove solo il quarto elemento dell'indirizzo può cambiare, ed è un valore compreso tra 0 e 255. Ad esempio, un intervallo valido è [160.97.62.1 - 160.98.62.12] .

Esercizio 3.

Si vuole realizzare uno script perl che imiti il comando `wc`. In particolare, lo script deve implementare un comando `wc [opzioni] file` in grado di stampare sullo standard output un conteggio delle *linee* o *parole* presenti su un *file* di testo specificato come parametro. In particolare, lo script wc accetta come *opzioni* le stringhe:

(1) `-l`
(2) `-w`
(3) `-w="value"`

Se presente l'opzione `-l` lo script restituisce in output in numero di linee presenti nel `file`. Se presente l'opzione `-w` lo script restituisce in output in numero di parole presenti nel `file`. Inoltre, l'opzione `-w` può essere associata ad un valore stringa (`-w="value"`). In tal caso, il comando restituisce il *numero di parole* presenti nel `file` corrispondenti al *valore* (`value`) indicato. Se *nessuna* delle opzioni è presente, lo script si comporta come nel caso (2). Si utilizzi come testo di prova il testo *i_promes.txt*.

Esercizio 4.

Il file `output.log` (del quale è possibile ottenere una copia di prova da http://www.gibbi.com/os) contiene al suo interno sequenze del tipo:

```
Jun 19 12:23:01 iskiros CRON[28101]: (pam_unix) session closed for user root
Jun 19 12:23:01 iskiros CRON[28099]: (pam_unix) session closed for user root
Jun 19 12:24:01 iskiros CRON[28105]: (pam_unix) session opened for user root by (uid=0)

Jun 19 12:30:27 iskiros sshd[28211]: (pam_unix) authentication failure; logname= uid=0
euid=0 tty=ssh ruser= rhost=160.97.63.83  user=martello

Jun 19 12:30:29 iskiros sshd[28211]: Failed password for alessandra from 160.97.63.83
port 39946 ssh2
Jun 19 12:30:33 iskiros sshd[28211]: Accepted password for alessandra from 160.97.63.83
port 39946 ssh2
Jun 23 22:06:27 iskiros sshd[3601]: Invalid user postgres from 210.75.200.9
Jun 23 22:06:27 iskiros sshd[3601]: (pam_unix) check pass; user unknown
Jun 23 22:06:27 iskiros sshd[3601]: (pam_unix) authentication failure; logname= uid=0
euid=0 tty=ssh ruser= rhost=210.75.200.9 user=panetta

Jun 23 22:06:28 iskiros sshd[3601]: Failed password for invalid user postgres from
210.75.200.9 port 45187 ssh2

Jun 25 18:14:01 iskiros CRON[27477]: (pam_unix) session opened for user root by (uid=0)
Jun 25 18:14:01 iskiros CRON[27479]: (pam_unix) session opened for user root by (uid=0)
Jun 25 18:14:01 iskiros CRON[27475]: (pam_unix) session closed for user root
Jun 25 18:14:02 iskiros CRON[27479]: (pam_unix) session closed for user root

Jun 26 07:05:01 iskiros CRON[8305]: (pam_unix) session opened for user root by (uid=0)
Jun 26 07:05:01 iskiros CRON[8307]: (pam_unix) session opened for user root by (uid=0)
```

in cui sono memorizzati eventi inerenti agli accessi effettuati sul sistema. In particolare, i tentativi di accesso non andati a buon fine sono segnalati come su righe contenenti la stringa "`authentication failure`" e l'utente relativo è indicato mediante la stringa "`user=nome_utente`". Ogni evento viene registrato su un'unica linea: la prima voce della linea rappresenta la data dell'evento nel formato indicato (Prime tre lettere del mese, giorno del mese, ore, minuti, secondi).

Il file `passwd` (della quale una copia di prova è reperibile sull'indirizzo di cui sopra) contiene al suo interno sequenze del tipo:

```
root:x:0:0:/root:/bin/bash
leone:x:500:500:/home/leone:/bin/tcsh
perri:x:501:500:/home/perri:/bin/tcsh
ianni:x:502:500:/home/ianni:/bin/bash
galizia:x:503:500:/home/galizia:/bin/bash

grillo:x:2005:2007:/home/grillo:/bin/bash
iapello:x:2006:2007:/home/iapello:/bin/bash
dig:x:2007:501:/home/dig:/bin/bash
```

in cui sono memorizzate informazioni relative agli account degli utenti del sistema, separate dal carattere ':' . Il primo campo rappresenta il nome dell'utente, il terzo indica lo `user-id` dell'utente e l'ultimo la `shell` ad esso associata.

Punto 1)

Si scriva uno script Perl `disable.pl` che riceve da linea di comando i file `output.log` e `passwd`. Lo script deve esaminare il file `output.log` per individuare i tentativi di accesso fallimentari. Nello specifico, il programma deve essere in grado di individuare il nome degli utenti che hanno fallito almeno due accessi nello stesso giorno; si considerino significativi solo il mese ed il giorno riportati (che corrispondono alle prime due voci di ogni rigo nel file `output.log`).

Il programma deve quindi disabilitare gli utenti così individuati modificando opportunamente il file `passwd`, sostituendo la shell assegnata all'utente con lo script `alert.sh` che si suppone già trovarsi nella directory `/bin`.

Lo script deve infine **aggiornare** un file `disabled.txt` contenente le informazioni relative a ciascuna coppia `<nome_utente,shell_associata>` relativa agli utenti disabilitati (si scelga un formato a piacere per il contenuto del suddetto file).

Punto 2)

Si scriva uno script perl `enable.pl` che riceve da linea di comando lo `user-id` (attenzione: lo user-id, NON il nome testuale) di un utente da riabilitare ed i file `disabled.txt` e `passwd`. Lo script, in base alle informazioni contenute nei file `passwd` e `disabled.txt`, deve riattivare l'accesso all'utente corrispondente alla `user-id` fornita, ripristinando la sua `shell` originaria.

NOTA BENE: L'esempio riporta un caso di uso pratico reale, dove al posto dei file dimostrativi messi a disposizione si può immaginare di lavorare sui file reali `/etc/passwd` e `/var/log/auth.log`.

*E' tuttavia è **sconsigliatissimo** disabilitare l'account di un utente in seguito a una serie di login falliti (Si pensi a un utente malintenzionato che usi questo sistema per forzare la disabilitazione dell'account di un utente legittimo, o addirittura dell'account root, o di tutti gli account di un dato server).*