3R-IN3 – Systèmes d’exploitation

TP3

Xavier Hilaire
x.hilaire@esiee.fr

31 mars 2025

1 Mélange (*contrôle 2020-2021*)

La variable spéciale `RANDOM` du `bash` renvoie un entier aléatoire entre 0 et 32767 à chaque fois qu’elle est évaluée. Exemple :

```
$ echo $RANDOM
17506
$ echo $RANDOM
2239
$
```

Vous disposez d’un `fichier` texte, dont vous pourrez supposer qu’il contient moins de 32767 lignes, et vous voudriez écrire une fonction ou un script `desordre` tel que

```
cat fichier | desordre
```

produise une version « mélangée » du `fichier`, c’est-à-dire que chacune de ses lignes continue à être affichée exactement 1 fois, mais à une nouvelle position imprévisible. En d’autres termes, votre script doit produire le même résultat que la commande `sort -R` sans l’utiliser, et vous n’avez pas le droit de créer de fichier, même temporaire.

Écrire ce script ou cette fonction. Le tester sur un fichier de votre choix. Indications : `sed`, `sort`, + 1 boucle `while`.

Sol.:

```
#!/bin/sh
#
while read ligne; do
    printf "$RANDOM:%s\n" "$ligne"
done | sort -n -k1 -t: | sed -E "s/^[0-9]+://"
```

2 Historique

On voudrait conserver l’historique des impressions demandées par les utilisateurs sur une même machine Linux. Pour cela, on maintient à jour un fichier `print_usage` que l’on archive hebdomadairement, et dont la forme générale est la suivante :

```
joe/lun. 15 mai 2023 09:31:00 CEST/58
hilairex/jeu. 01 juin 2023 17:15:44 CEST/482
```

Le caractère séparateur est '/', le premier champ est le login, le deuxième la date à laquelle la dernière impression a été demandée, et le troisième le nombre total de pages cumulées que l'utilisateur a imprimées depuis la création du fichier.

Ecrire un script `print_log login N`, acceptant un `login` et un nombre `N` de pages en arguments, qui fera les choses suivantes :

- Si le login spécifié est inconnu sur la machine (ce que vous pourrez vérifier en interrogeant la commande `id`), le script écrit un message d'erreur sur sa sortie d'erreur standard, et sort immédiatement avec le code 1
- Si le login spécifié est valide, mais ne figure pas dans le fichier `print_usage`, alors le script crée une nouvelle ligne avec le nombre spécifié comme troisième champ
- Si le login spécifié est valide, et qu'une ligne correspondante figure dans le fichier `print_usage`, alors le script met à jour le 3ème champ en lui ajoutant `N`, **et le 2ème en le remplaçant par le texte produit par la commande `date`**.

Indication : lorsqu'elle est utilisée conjointement avec la commande `s` (substitute) et un fichier, l'option `-i` de `sed` modifie directement et de manière permanente, le fichier en question. Consulter le manuel.

Sol.:

```
#!/bin/sh

FIC=print_usage

if id "$1" > /dev/null 2>&1; then # le login est valide
    now=$(date)
    if a=$(grep "^$1/" $FIC); then # utilisateur déjà connu
        n=${a##*/} # nombre courant de pages
        y=$2 # nombre de pages à ajouter
        sed -i "s=~$1.*\#$=$1/$now/$((n+y))=" $FIC
    else
        # utilisateur inconnu => ajouter une ligne
        echo "$1/$now/$2" >> $FIC
    fi
else
    echo "Login inexistant" 1>&2
    exit 1
fi
```

3 Filtre anti-spam

Cet exercice a pour but de vous faire utiliser les commandes `sed` et `grep`, et les regex étendues, pour détecter certaines anomalies susceptibles d'apparaître dans des mails indésirables, en analysant leur entête SMTP.

Commencez par récupérer le fichier `spam.eml` sur votre machine à l'aide de la commande suivante

```
curl -o spam.eml https://perso.esiee.fr/~hilairex/3R-IN3/spam.eml
```

3.1 Limitation à l'entête SMTP

L'entête SMTP consiste en l'ensemble des premières lignes de l'email, juste avant la ligne vide, qui marque sa fin. Ecrire une fonction `entete()` qui, à partir du nom de fichier qu'elle

recevra en unique argument, affichera seulement son entête SMTP. Aide : `head -n` et `grep`

Vous aurez intérêt à utiliser cette fonction dans toutes les fonctions suivantes. Si vous ne la trouvez pas, substituez-lui une commade `cat` (qui affichera la totalité du fichier)

Sol.:

```
## Fonction entete
# On commence par chercher les lignes *vides* avec grep. La première
# d'entre elles (-m 1) est celle qui marque la fin de l'entête. Au
# lieu d'afficher la ligne, on affiche son numéro (-n), qui est suivi
# d'un ':' indésirable, que l'on élimine (sed)
# On récupère ce nombre affiché dans la variable a
# Il suffit ensuite de relancer head -n $a pour n'afficher que les
# que les premières lignes du fichier
#

function entete() {
a=$(grep -n '^$' -m 1 "$1" | sed 's/://g')
head -n $a "$1"
}
```

Sol.:

3.2 Champ sujet

Ecrivez une fonction `sujet()` qui analyse le champ Subject de l'entête, et renvoie 1 dès que l'un au moins des cas suivants se produit (sinon, renvoie 0) :

- Le champ ne contient aucune lettre minuscule
- Le champ contient au moins deux points d'exclamation (pas nécessairement consécutifs)

Vérifier qu'elle fonctionne bien.

Sol.:

```
# Fonction sujet
#
# On doit d'abord isoler la bonne ligne avec grep et head. Détecter les '!'
# peut se faire avec expr et des chaînes arbitraires entre eux.
# Pour les majuscules, le test équivaut tout simplement a regarder s'il
# existe une seule lettre minuscule
#
function sujet () {
    local ligne="$(grep -m 1 -a ^Subject:)"
    local r=0

    if expr "$ligne" : '^.*!.*!.*$' >/dev/null; then
r=1
    elif ! expr "$ligne" : ".*[a-z].*" >/dev/null; then
r=1
    fi
}
```

```

    return $r
}

```

3.3 Champ expéditeur

Proposer une fonction `from()` qui analyse le champ `From` de l'email et renvoie 2 lorsque le domaine suivant l'arobase est invalide, sinon 0.

Vous pourrez pour cela interroger la commande `whois` avec les deux derniers mots seulement du domaine extrait (ex : pour `edu.esiee.fr`, vous appellerez `whois esiee.fr` et non `whois edu.esiee.fr`. Vous pourrez considérer que le test est validé lorsque `whois` produit à la fois :

- une ligne de la forme `domain: ledomaine.fr`
- et une autre de la forme `status: ACTIVE`

Sol.:

```

# Fonction from
#
# Comme pour sujet, on s'appuie sur les bracket <> et l'arobase pour extraire
# le nom de domaine. Comme le code d'erreur de nslookup est 0 qu'il trouve
# ou non un nom de domaine, il faut chercher s'il écrit "can't find" ou
# non pour conclure.
#
function from () {
    local x='grep -m1 ^From: | sed -r "s/^From:[^@]*@([^>]*).*$/\1/'
    local y=$x

    # raccourcir le domaine à machin.truc
    while ! test "${y#*}." = "$y"; do
        x=$y
        y=${y#*}
    done

    # valider avec whois : vérifier qu'on ait bien les 2 lignes voulues
    # par grep
    nlignes=$(whois $x | grep -E "^domain:.*$y|^status:.*ACTIVE" | wc -l)
    test $nlignes -eq 2 && return 0
    return 2
}

```

3.4 Antidatation

On voudrait vérifier que la date à laquelle l'entête SMTP prétend que le mail a été envoyé ne se situe pas dans le futur. Pour cela, la seule méthode possible est de comparer ce qu'affirme la ligne `Date:` de l'entête, à la date courante, que vous pouvez obtenir par la commande `date`

3.4.1 Fonction auxiliaire

Que fait la fonction suivante ? Examiner ce que produit la commande `date` et donner une commande qui relie `mois` et `date` judicieusement.

```

function mois () {

```

```

local liste="Jan/01:Feb/02:Mar/03:Apr/04:May/05:Jun/06:Jul/07:Aug/08:"\
"Sep/09:Oct/10:Nov/11:Dec/12:"
local arg="$1"

while test "$liste"; do
    tete=${liste%:*}
    mois=${tete%/*}
    num=${tete#*/}
    arg='echo "$arg" | sed "s/$mois/$num/'
    liste=${liste#*:}
done

echo $arg
}

```

3.4.2 Fonction principale

Proposer une fonction `antidate()`, qui utilise la fonction précédente, et renvoie 3 si le mail est antdaté, sinon 0.

Précisions et indications :

- En toute rigueur, il faudrait aussi inclure l'heure (avec son fuseau horaire) comme critère supplémentaire de discrimination au cas où les dates font référence au même jour. Vous pourrez passer outre cette précaution.
- La commande `date` est sensible à la variable d'environnement `LANG`
- Les dates sous forme alphanumérique, telles qu'elles sont données, ne sont pas comparables entre elles

Sol.:

```

function antidate () {
    local datemail='grep -m1 ^Date: | sed -r "s/^[^0-9]*([ ]*) ([ ]*) ([ ]*).*$/\3\2\1/'
    local date='LANG=C date -R | sed -r "s/^[^0-9]*([ ]*) ([ ]*) ([ ]*).*$/\3\2\1/'

    local dm='mois "$datemail"'
    local dr='mois "$date"'

    [ $dm -gt $dr ] && return 3
    return 0
}

```

4 Tri à partir de dates

La commande `last` affiche qui s'est logé sur la machine qui l'exécute, depuis un certain temps. Cette sortie est toujours triée par ordre chronologique. Mais lorsqu'on concatène deux sorties de `last`, obtenues sur des machines différentes, le résultat n'est plus forcément trié.

1. A l'aide d'une boucle et de `curl`, récupérer les 4 fichiers `last1` à `last4` dont l' URL générique est `https://perso.esiee.fr/~hilairex/3R-IN3/last$i` pour `i` variant de 1 à 4.
2. Ecrire un script (ou une fonction) qui fusionnera les sorties de deux commandes `last`, que vous donnerez sous forme de fichiers, en une seule sortie triée par date seulement (inutile de prendre en compte l'heure). Le tester sur deux des fichiers précédents.

Sol.: