TP OS n°5

Sommaire

Les objectifs de ce tp sont d'être capable, en utilisant des commandes de base sous GNU/-Linux, de grouper des commandes et d'utiliser les caractères spéciaux avec celles-ci.

Introduction

Environnement de travail

Il vous faut ouvrir une **session** sur votre poste de travail. Vous pouvez utiliser soit le mode console soit l'interface graphique. Dans les deux cas, vous allez travailler **« en ligne de commande »**.

Pour ce TP, il vous faudra préalablement créer un répertoire de travail `tpos4` dans `$HOME/tpos` en tapant la commande suivante :

```
$ mkdir -p $HOME/tpos/tpos5
```

Pour se déplacer dans l'arborescence de travail, il faut taper la commande suivante :

```
$ cd $HOME/tpos/tpos5
```

Groupement de commandes

Il existe plusieurs modes d'exécution :

```
cmd   // exécute la commande cmd
cmd & // exécute la commande cmd en tâche de fond
! cmd // inverse le code retour de la commande cmd (il y a un espace entre ! et cmd)
(cmd) // exécute la commande cmd dans un sous-shell
```

Il est possible de grouper plusieurs commandes :

```
cmd1 ; cmd2 // exécution séquentielle de cmd1 puis cmd2
cmd1 | cmd2 // tube (pipe) entre cmd1 et cmd2
cmd1 && cmd2 // si cmd1 retourne VRAI alors cmd2 sera exécuté
cmd1 || cmd2 // si cmd1 retourne FAUX alors cmd2 sera exécuté
```

Tous les processus se terminant renvoie un **code de retour** au *shell*. Ce code de retour est accessible par la variable **`$?`** et traduit (le plus souvent) l'état de l'exécution du programme. On utilise un programme pour remplir une tâche (processus) et celui-ci nous donne un rapport booléen par le code retour : VRAI (la tâche a été accomplie avec succès) et FAUX (la tâche a rencontré une erreur). Au minimum, le code de retour sera `0` ou `1`, mais dans le cas d'une autre valeur numérique, il pourra aussi traduire un type d'erreur.

Caractères spéciaux

Les caractères spéciaux ou génériques (*wildcard characters*) permettent de désigner un ensemble d'objet et notamment un ensemble de noms de fichiers (le caractère `*` étant le plus connu et le plus utilisé).

Ils peuvent aussi désigner un ensemble de chaînes de caractères. On parle alors d'**expressions rationnelles** (ou **expressions régulières**) qui s'appliquent aux commandes d'édition (`vi`, `sed`, ...) ou à des filtres (`grep`, `egrep`, `awk`, ...).

Une expression rationnelle (ou expression régulière) est une chaîne de caractères que l'on appelle parfois un motif et qui décrit un ensemble de chaînes de caractères possibles selon une syntaxe précise. Elles sont notamment aujourd'hui utilisées dans l'édition et le contrôle de texte. En savoir plus : `$ man 7 regex`

Les caractères associés aux noms de fichier sont interprétés par le shell avant le lancement de la commande :

```
*   désigne toutes les chaînes de caractères (y compris la chaîne vide)
?   désigne un caractère quelconque
[...]     désigne un caractère quelconque appartenant à la liste
[!...]    désigne une liste de caractères à exclure
{...,...} désigne une liste de caractères (une chaîne)
```

Les caractères associés aux expressions régulières :

```
.  désigne un caractère
*  remplace zéro fois ou n fois le caractère qui le précède
\+   remplace 1 fois ou n fois le caractère qui le précède
\?   remplace 0 zéro fois ou 1 fois le caratcère qui le précède
\b   désigne la chaîne vide (en début ou en fin de ligne)
[...]   désigne un caractère quelconque appartenant à la liste
^  désigne le début de la ligne
$  désigne la fin de la ligne
[^...]  désigne une liste de caractères à exclure
\{m\}   désigne un nombre exact m d'occurences d'un caractère
\{m,\}  désigne un nombre minimum m d'occurences d'un caractère
\{m,n\} désigne un nombre d'occurences d'un caractère compris entre un min m et un max n
\(...\) désigne une chaîne de caractère ou une expression régulière
\|   désigne une alternative
\<mot\> délimitation d'un mot
```

Il est possible d'annuler l'interprétation d'un caractère spécial ou de contrôle de trois manières en utilisant des caractères de protection :

```
\  : l'antislash annule la signification du caractère suivant
'...'   : les simples quotes annulent tous les caractères
"..."   : les doubles quotes annulent tous les caractères sauf ‘, \ et $
```

Manipulations

Étape n°1 : Groupement de commandes

Déterminons la signification de la valeur du code de retour d'une commande

```
$ ls ; echo $?
$ ! ls ; echo $?
$ rm abc* ; echo $?
```

Groupons des commandes avec `||`

```
// Le groupement || est notamment adapté à l'envoi conditionné de messages d'erreurs :
$ rm fff || echo "Houston, on a un problème !"
$ ls || echo "Houston, on a un problème !"
```

Groupons des commandes avec `&&`

```
// Le groupement && est notamment adapté à l'exécution d'un programme (cmd2) conditionné par
    la bonne exécution d'un autre programme (cmd1) :
$ ls *.txt && rm -f *.txt
$ ls *.log && rm -f *.log
```


Etant donné que les groupements `||` et `&&` permettent des exécutions conditionnées de commandes, il devient intéressant de présenter la commande `test`. Cette commande permet de réaliser de nombreux tests et de retourner le résultat du test sous forme d'un code retour (`$?`) :

```
$ touch test.log # crée un fichier vide

$ test -s test.log || echo "le fichier est vide"
$ test -e test.log && echo "le fichier existe"

$ help test
```

Étape n°2 : Les commandes grep, sed et awk

`grep`, `egrep`, `fgrep` permettent d'afficher les lignes correspondant à un motif donné. C'est l'une des commandes les plus utilisées (notamment dans des tubes) pour des recherches dans du texte.

`grep` peut utiliser des classes de caractères prédéfinies comme :

```
[:digit:] (chiffres), [:lower:] (minuscules), [:print:] (affichables), [:punct:] (
   ponctuation), [:space:] (espace), [:upper:] (majuscules), et [:xdigit:] (chiffres
   hexadécimaux).
Par exemple, [[:alnum:]] correspond à [0-9A-Za-z].
```

`sed` est un éditeur ligne non interactif. Il reçoit du texte en entrée, que ce soit à partir de stdin ou d'un fichier, réalise certaines opérations sur les lignes spécifiées de l'entrée, une ligne à la fois, puis sort le résultat vers *stdout* ou vers un fichier. A l'intérieur d'un script *shell*, sed est habituellement un des différents outils composant un tube. De toutes les opérations de la boîte à outils `sed`, on utilise principalement : `printing` (affichage vers *stdout*), `deletion` (suppression) et `substitution` (substitution).

```
// quelques exemples de sed :
1d : supprime la première ligne de l'entrée.
/^$/d : supprime toutes les lignes vides.
/Linux/p : affiche seulement les lignes contenant Linux
s/Windows/Linux/ : substitue Linux à chaque première instance de Windows
s/Windows/Linux/g : substitue Linux à chaque instance de Windows
s/ *$// : supprime tous les espaces à la fin de toutes les lignes.
s/00*/0/g : compresse toutes les séquences consécutives de zéros en un seul zéro.
/Windows/d : supprime toutes les lignes contenant Windows.
s/Windows//g : supprime toutes les instances de Windows, en laissant le reste de la ligne
   intact.
```

`awk` est un langage d'examen et de traitement de motifs. `awk` possède un langage de manipulation de texte plein de fonctionnalités avec une syntaxe proche du C. `awk` casse chaque ligne d'entrée en champs. Par défaut, un champ est une chaîne de caractères consécutifs délimités par des espaces (bien qu'il existe des options pour changer le délimiteur). `awk` analyse et opère sur chaque champ, ce qui le rend idéal pour gérer des fichiers texte structurés, particulièrement des tableaux, des données organisées en ensembles cohérents, tels que des lignes et des colonnes.

```
// Taille des partitions montés :
$ df | sed 1d | awk '{print $1 " = " $2}'
/dev/sda5 = 12G
/dev/sda7 = 34G
/dev/sda1 = 100M
/dev/sda2 = 49G
/dev/sda4 = 51G
```



```
/dev/sdb7 = 203G
/dev/sdb1 = 11G
/dev/sdb6 = 16G
/dev/sdc1 = 932G

// Espace disponible sur les partitions montés :
$ df | sed 1d | awk '{print $1 " = " $4}'
/dev/sda5 = 912M
...
```

Afficher toutes les lignes contenant au moins une majuscule :

```
$ getent passwd | grep '[A-Z]'
$ getent passwd | grep '[[:upper:]]'
```

Afficher toutes les lignes commençant par la lettre `a` :

```
$ getent passwd | grep '^[a]'
```

Afficher toutes les lignes contenant `bash` :

```
$ getent passwd | grep bash
```

Remplacer le *shell* `bash` par le *shell* `csh` pour tous les utilisateurs :

```
$ getent passwd | grep bash | sed 's/bash/csh/g'
```

Afficher toutes les lignes contenant au moins une occurrence de `bash` ou `sh` :

```
$ getent passwd | grep '\(bash\|sh\)\+'
```

Exploiter un motif :

```
$ echo "thierry.vaira@orange.fr" | sed 's/\(.*\)\@\(.*\)/nom:\1 domain:\2/'
```

Questions de révision

L'idée de base des questions de révision est de vous donner une chance de voir si vous avez identifié et compris les points clés de ce TP.

Question 1. Quel est l'intérêt de grouper des commandes ?

Question 2. Quel est le type des données composant les flux d'entrée et de sortie d'une commande ?

Question 3. Quelle est le différence entre une redirection d'E/S (par exemple `>`) et un tube (*pipe*) ?

Question 4. Quel est le délimiteur par défaut sur la ligne de commande ?

Question 5. Peut-on utiliser le caractère `*` dans un nom de fichier ?

Question 6. Que signifie pour le *shell* une valeur `0` renvoyée (code de retour) par une commande ? Et un code de retour égal à `1` ?

Question 7. Qu'est-ce qu'une expression rationnelle ?

Question 8. Donner la signification de ce motif "$([0-9]\{1,3\}\.)\{3\}[0-9]\{1,3\}$" ? A quoi peut-il servir ?

Travail demandé

Exercice 1 : groupement de commandes

Question 9. Que fait cette commande ?

```
$ getent passwd | grep $USER | cut -d: -f1 > data.txt
```

Question 10. Que fait cette commande ?

```
$ nom_commande 2>/dev/null && echo "ok" || echo "ko"
```

Question 11. Que fait cette commande ?

```
$ getent passwd | awk -F: '$3 >500 {print $1}'
```

Question 12. Que fait cette commande ?

```
# ifconfig | sed -e "s/^ *//g" | grep -Eo "([0-9]{1,3}\.){3}[0-9]{1,3}"
```

Question 13. Modifier le motif précédent pour filtrer toutes les adresses MAC renvoyées par la commande `ifconfig` ? Donner la commande.

Remarque : Une adresse MAC (Media Access Control) est un identifiant physique mondialement unique et stocké dans une carte réseau. Une adresse MAC est constituée de 48 bits (6 octets) et est généralement représentée sous la forme hexadécimale en séparant les octets par un double point (:) ou un tiret (-). Par exemple `5E:FF:56:A2:AF:15`.

Exercice 2 : filtrer une liste de fichiers

L'objectif de cet exercice est de manipuler les caractères spéciaux.

Créer avec la commande `touch` les fichiers suivants :

```
$ touch abc.s codage codage.c fichier.txt texte
```

Question 14. À quels fichiers correspondent les commandes ci-dessous ?

Commande	Fichier(s) filtré(s)
`$ ls *`	
`$ ls *.*`	
`$ ls *.?`	
`$ ls?.?`	
`$ ls f*`	
`$ ls a?c*`	
`$ ls *[ac]*`	
`$ ls [^a]*`	
`$ ls *.???`	
`$ ls *{abc,cod}*`	
`$ ls [a-c]*`	
`$ ls *.{c,txt}`	

