Laboratoire de téléinformatique
Linux Avancé

Objectif

Le but de ce laboratoire est de comprendre le fonctionnement de certains mécanismes et de certaines fonctionnalités du système d'exploitation Linux.

Introduction

Les manipulations proposées peuvent être effectuées sur d'autres distributions que celle installée au laboratoire.

Linux comporte une interface graphique assez efficace, par contre il est très conseillé de bien maîtriser le mode console (ligne de commande / mode texte). Un fois l'habitude prise, vous verrez que vous ne pourrez plus vous en passer car vous gagnerez énormément en efficacité.
Tout ce que va être dit dans ce laboratoire est valable aussi bien dans un terminal mode texte (tty1-6) que dans les terminaux graphique (xterm, …).

Le but de ce laboratoire est comme nous l'avons dit, de se familiariser aux possibilités offertes par ce système (édition et manipulation de fichiers, création de scripts, compilation,…).

Avant de poser des questions aux assistants, pensez à trouver vous-même la réponse à vos problèmes à l'aide de la commande `man` par exemple ou avec l'option `-help` que les commandes UNIX utilisent d'habitudes comme aide rapide. De plus le web regorge de documentation à propos de Linux, l'étudiant curieux a largement de quoi assouvir son intérêt.

Attention !

Comme déjà dit les manipulations proposées peuvent être effectuées sur d'autres distributions que celle installée au laboratoire. MAIS il se peut que certaines commandes ne fonctionnent pas. Ceci est du au fait que vous n'avez pas assez de privilèges pour l'exécution. En cas de problèmes n'hésitez pas à demander de l'aide aux assistants.

Laboratoire

Premièrement il faut vous mettre sur une station de travail (PC de MEM) et démarrer sous Linux :

```
Login :      labo
Password :   labo
```

A ce stade il faut ouvrir une console et nous pouvons commencer.

Une fois la console ouverte on se trouve dans un interpréteur de commandes, un *shell*. La plupart des distributions utilisent par défaut le shell *bash* (qui fait partie de la norme POSIX) mais ils en existent d'autres plus au moins compatibles comme par exemple zsh, ksh, csh, tcsh, ash.

L'intérêt d'avoir un shell plutôt qu'un autre dépend des fonctionnalités offertes par chacun. Chaque shell a sa propre syntaxe.

Voici une liste des commandes les plus utilisées permettant des opérations sur des fichiers et répertoires :

cd *rep*	Se déplace vers le répertoire *rep*
ls *rep*	Liste le contenu d'un répertoire (*-R* pour lister les autres répertoires rencontrés).
cp source destination	Copie un fichier *source* vers un fichier *destination* (*-R* pour un répertoire).
mv source destination	Bouge le contenu d'un fichier ou répertoire *source* vers un fichier ou répertoire de *destination* (utilisé aussi pour renommer un fichier ou répertoire).
mkdir *rep* :	Crée un répertoire.
rm fichier :	Efface un fichier (*-r* pour effacer un répertoire en entier).
ln source destination	Crée un lien *destination* qui va pointer sur *source* (pour un lien symbolique, il suffit d'ajouter l'option "-s")
touch fichier ou repertoire	Met à jour la date de modification du fichier, ou crée un fichier vide si le fichier n'existait pas.

cat fichier	Lit le contenu du fichier.
more fichier	Lit le contenu d'un fichier page par page. (Il doit lire l'intégralité du fichier avant de l'afficher).
less fichier	Comme "more" sauf qu'il n'est pas obligé de lire l'intégralité du fichier.
tail fichier	N'affiche que les dernières lignes d'un fichier (d'autres options permettent de spécifier combien de lignes afficher).
head fichier	Comme *tail*, mais affiche les N premières lignes d'un fichier (N=10 par défaut).
find répertoire options	Permet de rechercher des fichiers suivant une quantité incroyable de critères.

grep "chaine" fichier	Recherche l'occurence d'une chaine de caractères "chaine" dans un ou plusieurs fichiers.

Il y en a bien d'autres, mais à ce stade, il n'est pas encore nécessaire de les voir. Nous les découvrirons par la suite.

Rappelons que les noms des fichiers sont *case sensitive,* c'est à dire qu'ils tiennent compte des majuscules et des minuscules. Le nom d'un fichier peut contenir jusqu'à 255 caractères.

Pour accéder à un fichier comportant des espaces dans son nom, deux méthodes sont à disposition :

- placer le nom du fichier entre guillemets
- utiliser des backslash avant les espaces (le caractère *espace* est considéré comme un caractère spécial)

Exécuter les lignes suivantes et commenter les résultats :

```
touch "ceci est un test.txt"
ls ceci\ est\ un\ test.txt
```

Notons que les fichiers ou répertoires dont le nom commence par un point « . » sont cachés, c'est à dire qu'ils ne seront pas visibles avec la commande `ls`. Si on veut lister les fichiers cachés, il suffit de passer l'argument `-a` à `ls`.

Que font les lignes suivantes ? Commentez.

```
touch ".ceci est un test.txt"
ls
ls -a
```

Le shell Bash n'est pas capable d'interpréter les expressions régulières comme le font par exemples les commandes `sed` et `awk`. Par contre il est capable d'interpréter les caractères jokers comme :

- * correspond à aucun ou plusieurs caractères
- ? correspond à un caractère
- [a-Z] correspond à un ensemble de caractères
- [^a-Z] correspond à tous les caractères sauf ceux de cet ensemble

Que font les lignes suivantes ? Commentez.

```
ls /*t?/
ls -l /pr*c/s?s/n?t/ipv4/co*/*/forwarding
ls /[d-e]?[^v]/../home/../    (le .. indique le répertoire père)
```

Pour voir l'espace disque utilisé par vos fichiers ou répertoires, il faut utiliser la commande `du`.

Exécuter la ligne suivante et commenter les résultats :

```
du -sh .    (le . indique le répertoire courant)
```

Pour lister les processus tournant sur le système on utilise la commande `ps`. Les différents arguments permettent de donner plus de détails sur les processus.

Exécuter la ligne suivante et commenter les résultats :

```
ps aux
```

La commande `top` permet aussi de lister les processus, mais elle donne aussi d'autres informations comme la charge du système, …

Exécuter la ligne suivante et commenter les résultats :

```
top
```

Que fait l'option `-d 1` ?

Il est possible dans Bash de faire des alias pour les commandes les plus courantes. Ceci est effectué à l'aide de la commande `alias`.

Exécuter les lignes suivantes et commenter les résultats :

```
alias lf='ls -laF'
lf
```

Lorsqu'on travaille avec Linux pour la première fois on est un peut dépaysé car on ne retrouve plus les périphériques « à la Windows ». C'est à dire que le lecteur de disquettes, par exemple, n'est plus le lecteur A, le premier disque n'est plus le lecteur C,… Sous Linux on parle de montage de disque, car on doit *monter* le périphérique quelque part dans l'arborescence du système avant de l'utiliser. On peut ensuite y accéder simplement en allant dans le répertoire correspondant. Si on monte une disquette, un disque local, un disque distant (NFS) ou un lecteur cdrom, du point de vue du système de fichiers tout est transparent. Il est important de ne pas oublier de *démonter* les périphériques une fois plus utilisés.

Pour monter un périphérique on utilise la commande `mount`, pour démonter la commande `umount`.

1. Insérer une disquette dans le lecteur et exécuter les lignes suivantes. Commentez les résultats :

```
mount /dev/fd0 /mnt/floppy
ls /mnt/floppy
umount /dev/fd0   ou    umount /mnt/floppy
```

2. Insérer une clé USB dans le lecteur et exécuter les lignes suivantes. Commentez les résultats :

```
mount /dev/sda1 /mnt/usb
ls /mnt/usb
umount /dev/sda1  ou    umount /mnt/usb
```

Pour info : utilisez la commande « sudo » avec le mot de passe « labo ».

Linux est un OS multitâches et multi utilisateurs, il est évident qu'on puisse se *loguer* en même temps dans plusieurs consoles. Pour changer de console (appelées tty[1-12]), il suffit de taper Alt-F1, Alt-F2,… Si vous êtes dans une session Xwindows vous devez par contre taper Ctrl-Alt-F1,… puisque les touches Alt- sont déjà utilisées par le système de fenêtrage.

Essayer de changer de console, par exemple tty1. La console que Xwindows utilise est tty7 donc pour revenir dans X taper Alt-F7.

Linux nous offre la possibilité de nous connecter facilement sur une machine distante à l'aide de la commande ssh (Secure SHell).

Connectez-vous sur la machine d'un autre groupe et commenter les résultats :

```
ssh labo@IPdelautregroupe
ls
exit
```

Maintenant on va passer en revue deux éditeurs de texte *streaming*. Ces commandes sont `sed` et `awk`. Sed permet par exemple de remplacer ou effacer du texte dans un flux. Pour mieux comprendre, exécuter la ligne suivante. Pour plus de détails référez vous au manuel de sed.

```
echo "This is windows not Windows" | sed s/"windows"/"Linux"/g
```

Awk permet de reformater du texte (et d'un stream), par exemple :

```
echo "This is windows not Windows" | awk –F" " '{print $4}'
```

Quel est le résultat de la commande ?

Pour plus de détails référez vous au manuel de awk ou gawk.

Nous allons voir maintenant l'utilisation de l'éditeur de texte `vi`. Il existe sous Linux de nombreux autres éditeurs, partant de `ed` pour le plus simpliste à `jed` ou `emacs`. Alors pourquoi vi ? La réponse est simple, c'est un éditeur que vous retrouverez sur 99,99% des systèmes Unix. La plupart des serveurs Linux passent par l'édition de fichiers. Il est donc essentiel de maîtriser l'utilisation de vi pour être en mesure d'agir sur n'importe quel système Unix, Linux compris.

Vi possède deux modes de fonctionnement, contrairement aux éditeurs wysiwyg (what you see is what you get). Une fois lancé votre futur éditeur préféré (vi) vous vous trouvez en mode commande et vous pouvez ainsi ouvrir un nouveau fichier, quitter, effacer une ligne, vous déplacer dans le texte, copier, coller,… Cependant vous ne pouvez pas écrire ! Pour ce faire, il faut passer en mode insertion.
Une fois que vous avez terminé d'écrire, ne pas oublier d'appuyer sur la touche ESQ pour revenir en mode commande.

Mode insertion

> Les caractères tapés s'insèrent dans le texte du fichier édité. Un jeu réduit de commandes est disponible par l'intermédiaire de caractères de contrôle.
> Ce mode se termine en tapant <ESC>.

Mode commande

> Les caractères tapés sont considérés comme des commandes d'édition de texte.
> Attention : Les commandes sont *case sensitive*.

En annexe vous trouverez un aide-mémoire des commandes de vi. Vous pouvez l'utiliser comme référence dans le cas de ce laboratoire.

Étude de cas 1

Cet exemple de script utilise plusieurs notions apprises auparavant.
Créez un fichier vide de nom `myfile` à l'aide de la commande `touch`.
Ouvrez deux consoles Bash indépendantes.

Dans une console taper :

```
tail -f myfile
```

Que fait l'option `-f`

Dans l'autre console taper :

```
echo "" > myfile; (a=0;while [ $a -lt 10 ]; do echo -e \
"$a\tCiao";a=`expr $a + 1`;sleep 2; done >> myfile) & \
(a=0;while [ $a -lt 20 ]; do echo -e "$a\tSalut"; a=`expr \
$a + 1`;sleep 1; done >> myfile) & wait
```

Que fait le script? Quelle est sa particularité ? À quoi sert la commande `wait` ? À quoi sert la commande `&` ? À quoi sert la commande `\` ?

Étude de cas 2

Vous connaissez maintenant plusieurs commandes et leur utilisation. Avec celles que vous avez apprise lors du premier laboratoire Linux (Labo Linux de base) vous devez créer un script à l'aide de l'éditeur vi.

Ce script devra afficher :

- Le numéro de la version du noyau
- la vitesse du processeur en MHz
- la taille totale de la mémoire
- le nombre de processus total

Le format d'affichage sera le suivant :

```
Version du noyau :      2.0.7
Vitesse du proc :       1725.127 MHz
Taille Tot Mem :        516616 kB
Nombre de procs :       61
```

Étude de cas 2 - AIDE

Vous trouverez certaines informations dans les fichiers suivants (utiliser par exemple `cat`) :

```
/proc/version
/proc/cpuinfo
/proc/meminfo
```

Ces fichiers sont virtuels, leur taille est donc nulle. Ce sont des points de contact avec le noyau ; les informations qu'on y retrouve c'est le kernel qui nous les donne.

Vous trouverez d'autres informations avec les commandes :

```
top
awk
```

Dans votre script vous devez utiliser des pipes | et des variables. Les commandes suivantes affectent une variable et l'affichent :

```
MyVar=$(ps aux | grep ssh | awk '{print $2}')
echo "Voici ma variable $MyVar"
```