Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université de Carthage

Ecole Nationale d'Ingénieurs de Carthage

Fascicule de TP

Module : Atelier Systèmes d'Exploitation

Niveau : 2ème Année Ingénieur Informatique

Réalisé par

Mme Inès ACHOUR YAZIDI

Module : Atelier Systèmes d'exploitation
Nature : Travaux pratiques.
Volume horaire : 3 heures par semaine.
Public cible : 2ème année Ingénieur en Informatique.
Semestre : Premier
Prérequis : Aucun.

Modalité d'évaluation :

- Examen de TP.

Objectifs

Le module Atelier système d'exploitation est une matière axée sur la pratique pour que l'étudiant acquiert un savoir et un savoir-faire minimal et soit capable d'une part de tirer profit des différentes fonctionnalités d'un système d'exploitation et d'autre part d'installer et configurer un système d'exploitation. Afin de faire profiter les étudiants de la certification LPIC–1 (Linux Professional Institute) surtout que l'ENICarthage est un centre de certification, j'ai essayé d'adapter ce module aux différentes compétences du LPI 101.

Références

[1] Sébastien ROHAUT, « LINUX : Préparation à la certification LPIC-1 (LPI 101 LPI 102) », édition ENI. URL :

http://www.chercheinfo.com/uploads/1970-1df027e42c.pdf

[2] Afef ELLOUZE, Naïma HALOUANI ZGHAL, « Linux : préparation à la certification LPI 101 », Centre de Publication Universitaire, 2010.

[3] Mohamed Saïd OUERGHI, « Principes des systèmes d'exploitation », Centre de Publication Universitaire, 2003.

TP n°1 : Installation Linux

TP n°2 : Gestion des fichiers (1)

TP n°3 : Gestion des fichiers (2)

TP n°4 : Programmation Shell (1)

TP n°5 : Programmation Shell (2)

TP n°6 : Gestion des processus

TP n°7 : Gestion de l'architecture

TP n°8 : Divers : les droits avancés, partitionnement, les quotas et la gestion des paquetages

TP1 : Installation Linux

Objectif : Ce TP a pour but de déterminer le meilleur schéma de partitionnement possible, il convient pour toutes les distributions.

Enoncé : Vous disposez sur votre PC personnel d'un disque de 160 Go dont 40 sont déjà occupés par un autre système. Votre machine dispose de 2 Go de mémoire vive.

Il vous reste 120 Go d'espace disque. Comment pouvez-vous les répartir, sachant que vous voulez séparer vos données du système ?

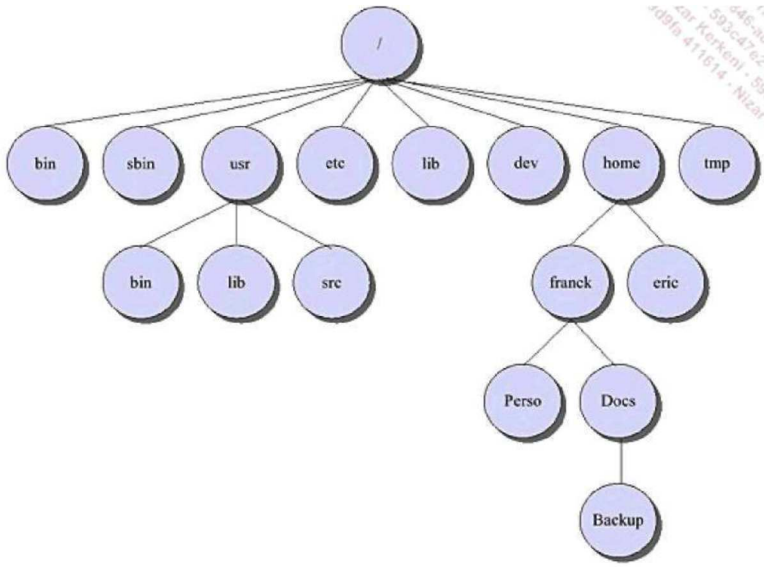
1. Quelle doit être la taille de la partition d'échange SWAP ?
2. Quelle place réserver au système / ?
3. Quelle place réserver au /home ?
4. Est-il utile de créer une partition étendue ?
5. Quel est le schéma final du disque ?

Complément du TP1

Le système de fichiers

Un système de fichiers, appelé communément File System ou FS, définit l'organisation des données sur un support de stockage, donc comment sont gérés et organisés les fichiers par le système d'exploitation.

Linux est, comme tout Unix, un système d'exploitation entièrement orienté fichier. Tout (ou presque) est représenté par un fichier, tant les données (fichiers de données de tout type comme une image ou un programme), que les périphériques (terminaux, souris, clavier, carte son, etc.) ou encore les moyens de communication (sockets, tubes nommés, etc.). On peut dire que le système de fichier est le cœur de tout système Unix.



Exemple d'arborescence Linux

Le système de fichiers de Linux est hiérarchique. Il décrit une arborescence de répertoires et de sous-répertoires, en partant d'un élément de base appelé la racine ou root directory.

/ représente la racine du système de fichiers. Ce répertoire est souvent appelé répertoire racine ou root. C'est le point de départ de toute arborescence.

/bin contient les programmes nécessaires pour démarrer le système en mode mono-utilisateur.

/boot contient les fichiers utiles pour le chargeur (ou boot loader). Ce répertoire ne contient que les fichiers nécessaires au boot.

/dev contient des fichiers spéciaux (device files) qui sont des points d'entrées vers des périphériques physiques. Exemple : le premier port série correspond au fichier /dev/cua0.

/etc contient les fichiers de configuration dont les divers programmes se servent pour avoir des informations sur la configuration du système.

/etc/X11 idem, mais spécifique au système Xwindow.

/home contient les données propres à chaque utilisateur. La structure interne du répertoire /home est laissé au bon vouloir de l'administrateur système. Dans la plupart des cas, vous êtes le seul à utiliser la machine et ce répertoire ne nécessite quasiment aucune modification.

/lib contient les bibliothèques (shared libraries) nécessaires aux programmes de démarrage (principalement des programmes placés dans /bin et /sbin).

/mnt contient des répertoires utilisés comme points de montage pour les périphériques en mode block. Habituellement, on y trouve cdrom et floppy, respectivement utilisés pour le lecteur de CD-Rom et le lecteur de disquettes. Une mauvaise habitude considere à créer un répertoire /cdrom à la racine, mais il est plutôt conseillé de créer un lien symbolique entre /mnt/cdrom et /cdrom par ln -s /mnt/cdrom /cdrom

/proc contient des informations sur l'état du système et les différents processus en fonction.

/sbin contient les programmes nécessaires au fonctionnement du système. Les commandes placées dans /sbin (et dans /bin) ne sont en général pas exécutées par les utilisateurs.

/tmp contient tous les fichiers temporaires générés par le système ou les processus en cours.

/usr contient une arborescence complète pour les données que les utilisateurs peuvent se partager. En principe, le répertoire /usr est le point de montage d'une partition ou d'un disque autre que celui utilisé pour le répertoire racine. Il est monté en lecture seule au démarrage et peut être partagé entre plusieurs machines.

/var contient des fichiers qui sont susceptibles de changer fréquemment, comme les journaux de l'activité du système (logs), les files d'attentes pour les impressions ou encore les crontabs des utilisateurs.

TP2 : Gestion des Fichiers

1. Quelles sont les informations que nous pouvons déduire de l'invite de commande?

2. À partir de votre répertoire personnel créez la structure suivante :

```
|---- dossier1
|       |---- dossier3
|---- dossier2
|       |---- dossier4
```

Utilisez les commandes **mkdir** (pour la création des répertoires) et la commande **cd** (pour le positionnement et le déplacement entre les dossiers).

3. En se positionnant dans votre répertoire personnel, enrichissez votre arborescence par la branche ci-dessous, et ce en utilisant une seule commande :

```
|---- dossier5
|       |---- dossier6
```

Utilisez la commande **mkdir** avec le paramètre -p :

4. Déplacez-vous dans le répertoire dossier1 avec un chemin absolu et créez le fichier fichier1 dans ce répertoire.
5. Copiez fichier1 dans le répertoire dossier3 avec un chemin relatif.
6. Déplacez-vous dans dossier2 en utilisant un chemin relatif, et copiez le fichier fichier1 de dossier3 sous le nom fichier2 là où vous êtes (dans dossier2) et dans dossier5.
7. Renommez et déplacez fichier2 en fichier3 dans le répertoire dossier3 (la commande **mv** déplace et renomme).
8. Supprimez fichier1 du répertoire dossier3 (la commande **rm** supprime le fichier).
9. Avec rmdir supprimez dossier4, puis dossier2 et tout son contenu. Est-ce possible ? Pourquoi ? Comment faire ?
10. Faites une copie fichier2 du dossier5 dans le dossier6 sans utiliser la commande cp (en utilisant les redirections)
11. Pour repérer les modifications apportées à votre arborescence, en vous positionnant dans votre répertoire personnel taper **ls** et chercher dans le **man** l'option qui vous permet d'afficher la liste des sous-répertoires.
12. Quel est le but de la commande **ls -l [a-z]*. ??[!0-9]** ?
13. Lister les entrées du répertoire /usr/bin dont le nom commence par la lettre m.
14. Lister les entrées du répertoire /usr/bin dont le nom commence par la lettre m et comporte exactement 3 caractères.
15. Lister les entrées du répertoire /usr/bin dont le nom commence par la lettre m et comporte au moins 3 caractères.
16. Lister les noms de fichiers comportant une étoile *.
17. En vous aidant du man de ls déterminer l'option qui vous permet d'afficher plus d'informations. Spécifier l'utilité de chaque colonne.
18. Pour chacune des propositions suivantes, calculez les droits en octal et spécifiez le

type du Fichier :
-rwx r—r-x
lr—rwx r—
prwx --- ---

19. Pour le dossier6 et le fichier2 du dossier5, quels sont les droits par défaut ? Peut-on-spécifier d'autres droits par défaut ?
20. Après création d'un fichier ou un répertoire peut-on changer les droits par défaut ?
21. Accorder au dossier9 exactement le droit de lecture aux autres.
22. Accorder au fichier9 le droit d'écriture au propriétaire.
23. Oter au dossier9 le droit d'exécution au groupe.
24. Citer les différents types de liens.

Complément de TP2

Créer des répertoires

La commande **mkdir** (*make directory*) permet de créer un ou plusieurs répertoires, ou une arborescence complète.

Par défaut la commande ne crée pas d'arborescence. Si vous passez comme arguments rep1/rep2 et que rep1 n'existe pas, la commande retourne une erreur. Dans ce cas, utilisez le paramètre -p.

```
mkdir [-p] rep1 [rep2] ... [repn]
```

```
$ mkdir Documents
$ mkdir Documents/Photos
```

Supprimer des répertoires

La commande **rmdir** (*remove directory*) supprime un ou plusieurs répertoires. Elle ne peut pas supprimer une arborescence. Si des fichiers sont encore présents dans le répertoire, la commande retourne une erreur. Le répertoire ne doit donc contenir ni fichiers ni répertoires et ceci même si les sous-répertoires sont eux-mêmes vides.

rmdir rep1 [rep2] ... [repn]

Il n'y a pas de paramètre -r (pour récursif) à la commande **rmdir**. Pour supprimer une arborescence vous devrez utiliser la commande **rm**.

```
$ rmdir Documents/
```

```
rmdir: Documents/: Le répertoire n'est pas vide.
```

```
$ rmdir Documents/Photos$
```

Copier des fichiers

La commande **cp** (copy) copie un ou plusieurs fichiers vers un autre fichier ou vers un répertoire.

cp fic1 [fic2 ... ficn] Destination

Dans le premier cas, fic1 est recopié en Destination. Si Destination existe, il est écrasé sans avertissement selon le paramètre passé et selon les droits. Dans le second cas, fic1, fic2 et ainsi de suite sont recopiés dans le répertoire Destination. Les chemins peuvent êtres absolus ou relatifs.

Liens

Création de lien symbolique :

```
ln -s fichier liensymbolique
```

Création de lien physique :

```
ln fichier lienphysique
```

La commande cd

Pour vous déplacer dans les répertoires, vous utilisez la commande **cd** (*change directory*). La commande **pwd** (*print working directory*) affiche le chemin complet du répertoire courant.
Si vous saisissez « **cd . »** , vous ne bougez pas. Le point sera très utile lorsque vous devrez spécifier des chemins explicites à des commandes situées dans le répertoire où vous êtes positionné.

Le **« cd .. »** remonte d'un niveau. Si vous étiez dans /home/user1, vous vous retrouvez dans home.
La commande **cd** sans argument permet de retourner directement dans son répertoire personnel.

Créer des fichiers vides : touch

Pour vos tests, vous pouvez avoir besoin de créer des fichiers vides. Avec **touch**.

Lister les fichiers et les répertoires : ls

La commande **ls** permet de lister le contenu d'un répertoire.

Supprimer des répertoires

La commande **rmdir** (*remove directory*) supprime un ou plusieurs répertoires. Elle ne peut pas supprimer une arborescence.

Déplacer et renommer un fichier

La commande **mv** (*move*) permet de déplacer, de renommer un fichier, ou les deux en même temps.

Supprimer un fichier ou une arborescence

La commande **rm** (*remove*) supprime un ou plusieurs fichiers, et éventuellement une arborescence complète, suivant les options. La suppression est définitive.

```
rm [Options] fic1 [fic2...]
```

TP3 : Gestion des Fichiers (2)

Exercice 1 :

Choisir la ou les bonnes réponses et justifier votre choix.

1. La commande `alias liste='ls>tmp'` :
   a) crée un alias liste qui exécutera ls >tmp
   b) crée un fichier tmp vide et un alias liste qui exécutera ls
   c) crée un fichier tmp qui contient liste=ls
2. Le fichier .profile permet de personnaliser :
   a) mon environnement Unix
   b) mon environnement XWindows
   c) mon profile d'utilisateur Netscape
3. La commande ls -la /usr/dup fournit le résultat ci-dessous :

```
drwxrwxr-- 3 u1  g1  512 Mar 6 17:36 .
drwxr-xr-x 3 bin sys 512 Mar 6 17:22 ..
-rwxr----- 1 u1  g1  180 Mar 6 17:36 f1
drwxr-xr-x 2 u1  g1   24 Mar 6 17:24 r1
```

Précisez pour chacune des commandes ci-après, si elle est autorisée ou non en donnant la raison :

- `who > /usr/dup/f1` par l'utilisateur u3 du groupe g1.
- `who > /usr/f1` par l'utilisateur u1 du groupe g1.

4. La commande ls $HOME/* affiche :
   a) le contenu de toutes les entrées du répertoire de connexion
   b) le contenu du répertoire de connexion
   c) l'entrée de nom * du répertoire de connexion

Exercice 2 :

Créer deux fichiers act1 et act2 contenant chacun votre position dans l'arborescence (pwd), votre nom de connexion (whoami) ainsi que votre identité complète (id) en respectant les consignes ci-après :

1. pour créer le fichier act1, commencer par créer trois fichiers séparés en n'utilisant que des redirections simples (>), puis concaténer tous les trois avec une nouvelle redirection simple pour créer ce fichier.
2. pour créer le fichier act2, utiliser les redirections en mode ajout (>>) pour créer le fichier et le compléter au fur et à mesure.

Vérifier à l'aide d'une commande de comparaison (diff et/ou cmp) que ces deux fichiers sont identiques.

Exercice 3 :

Affichez à l'écran les messages ci-dessous dans lesquels les textes entre guillemets doivent être remplacés par le contenu de variables et/ou le résultat de commandes

1. UNIX est un produit d'AT&T
   NB: le caractère «&» est un caractère spécial.

2. Mon répertoire de connexion $HOME est **"contenu d'une variable à mettre ici"**
3. Le code du caractere " est 34, celui de ' est 39 et celui de * est 42
4. La date est **"résultat de la commande à mettre ici"**
5. Le nombre d'utilisateurs connectés est : **"résultat de la commande à mettre ici"**

Existe-t-il une différence entre le résultat des deux commandes ci-après ? et si oui, laquelle ?

```
echo  "Je suis sous $HOME"
echo  'Je suis sous $HOME'
```

Complément de TP3

alias

Un alias est un raccourci d'une commande avec d'éventuels paramètres. Il se définit avec la commande **alias**. Utilisée seule elle liste les alias disponibles.

id

La commande **id** permet d'obtenir ces informations. En interne, le système travaille uniquement avec les UID et GID, et pas avec les noms eux-mêmes.

```
$ id
uid=1000(seb) gid=100(users) groupes=7(lp),16(dialout),33(video),100(users)
```

Comparaison de fichiers

Les deux commandes permettant de comparer le contenu de deux fichiers, ou d'un fichier et d'un flux sont les commandes **diff** et **cmp**.

a. diff

La commande **diff** indique les modifications à apporter aux deux fichiers en entrée pour que leur contenu soit identique.

```
diff [-b] [-e] fic1 fic2
```

b. cmp

La commande **cmp** compare les fichiers caractère par caractère. Par défaut la commande s'arrête dès la première différence rencontrée et indique la position de l'erreur.

```
cmp [-l] [-s] fic1 fic2
```

who

Affiche la liste des utilisateurs connectés au système.

whoami et who am i

whoami ne donne pas le même résultat que who am i si l'on est en cours de substitution d'utilisateur (su). Avec who am i, on obtient toujours l'identité de départ (par exemple, si l'utilisateur *gerard* adopte l'identité de *root* avec su, whoami retourne *root* tandis que who am i renvoie bien *gerard*).

TP4 : Programmation Shell (1)

Exercice 1 :

Ecrivez un shell script appelé « mon_script » qui affiche son nom, le nombre de paramètres fournis, la liste des paramètres, la valeur du premier paramètre, ainsi que les valeurs de deux variables : l'une initialisée et l'autre saisie.

Exercice 2 :

Ecrivez un script shell qui teste si le paramètre passé :

- Est un fichier
- Est un répertoire

Si aucun paramètre n'est passé, on affichera un message d'erreur indiquant le mode d'emploi du script.

Exercice 3 :

Ecrire un script Shell «**Compta**» permettant de compter le nombre de caractères ou de mots ou de lignes d'un fichier donné en argument. Le type d'éléments à compter est introduit interactivement par l'utilisateur.

Exercice 4 :

Ecrire un script shell qui calcule la somme de ses paramètres :

./somme 1 2 3 4 5

total = 15

Exercice 5 :

1. Ecrire un script shell qui affiche la nième ligne (premier paramètre du script) du fichier donné comme 2ième paramètre du script (proposer deux solutions).
2. Modifier le script précédent pour pouvoir accepter plus d'un fichier.

Exercice 6 :

1. Ecrire un script Shell **"traA"** permettant de transformer tous les caractères "a" en "A" des noms d'une série de fichiers passés en argument.
2. Modifier ce script pour pouvoir transformer toutes les chaînes de caractères "aba" en "Aba".

Exercice 7 :

1. Ecrire un script Shell "ListeFich" qui permet d'afficher toutes les informations des fichiers ordinaires d'un répertoire passé en argument. Traiter, par défaut, le cas du répertoire courant.
2. Modifier ce script pour n'afficher que les fichiers accessibles en lecture.
3. Modifier ce script pour n'afficher que les fichiers dont le nom est composé au plus de cinq caractères.
4. Afficher cette liste triée et en majuscule.
5. Afficher le nième fichier dans cette liste.

Complément du TP4

Paramètres de position

Les paramètres de position sont aussi des variables spéciales utilisées lors d'un passage de paramètres à un script.

Variable	Contenu
$0	Nom de la commande (du script).
$1-9	$1,$2,$3... Les neuf premiers paramètres passés au script.
$ #	Nombre total de paramètres passés au script.
$*	Liste de tous les paramètres au format "$1 $2 $3 . . . " .
$@	Liste des paramètres sous forme d'éléments distincts "$1 " "$2" "$3"

if ... then ... else

La structure **if then else fi** est une structure de contrôle conditionnelle.

```
if <commandes_condition>
then
<commandes exécutées si condition réalisée>
else
<commandes exécutées si dernière condition pas réalisée>
fi
```

Vous pouvez aussi préciser **elif**, en fait un else if. Si la dernière condition n'est pas réalisée, on en teste une nouvelle.

Choix multiples case

La commande **case ... esac** permet de vérifier le contenu d'une variable ou d'un résultat de manière multiple.

```
case Valeur in
Modele1) Commandes ;;
Modele2) Commandes ;;
*) action_defaut ;;
esac
```

Le modèle est soit un simple texte, soit composé de caractères spéciaux. Chaque bloc de commandes lié au modèle doit se terminer par deux points-virgules. Dès que le modèle est vérifié, le bloc de commandes correspondant est exécuté. L'étoile en dernière position (chaîne variable) est l'action par défaut si aucun critère n'est vérifié. Elle est facultative.

Les boucles

Elles permettent la répétition d'un bloc de commandes soit un nombre limité de fois, soit conditionnellement. Toutes les commandes à exécuter dans une boucle se placent entre les commandes **do** et **done**.

a. Boucle for

La boucle **for** ne se base pas sur une quelconque incrémentation de valeur mais sur une liste de valeurs, de fichiers ...

```
for var in liste
do
commandes à exécuter
done
```

La liste représente un certain nombre d'éléments qui seront successivement attribués à var.

- Avec une variable (exemple : $*)
- Liste implicite (Si vous ne précisez aucune liste à for, alors c'est la liste des paramètres qui est implicite).
- Avec une liste d'éléments explicite (Chaque élément situé après le « in » sera utilisé pour chaque itération de la boucle, l'un après l'autre).
- Avec des critères de recherche sur nom de fichiers (Si un ou plusieurs éléments de la liste correspond à un fichier ou à un motif de fichiers présents à la position actuelle de l'arborescence, la boucle for considère l'élément comme un nom de fichier).
- Avec une substitution de commande (Toute commande produisant une liste de valeurs peut être placée à la suite du « in »).

b. Boucle while

La commande **while** permet une boucle conditionnelle « tant que ». Tant que la condition est réalisée, le bloc de commande est exécuté. On sort si la condition n'est plus valable.

```
while condition

do

commandes

done
```

c. Boucle until

La commande **until** permet une boucle conditionnelle « jusqu'à ». Dès que la condition est réalisée, on sort de la boucle.

```
until condition

do

commandes

done
```

d. Boucle select

La commande **select** permet de créer des menus simples, avec sélection par numéro. La saisie s'effectue au clavier avec le prompt de la variable PS3. Si la

valeur saisie est incorrecte, une boucle s'effectue et le menu s'affiche à nouveau. Pour sortir d'un select il faut utiliser un **break**.

```
select variable in liste_contenu

do

traitement

done
```

Si in liste_contenu n'est pas précisé, ce sont les paramètres de position qui sont utilisés et affichés.

sort [options] [fichier(s)]

Cette commande lit ses données, ligne par ligne, par le canal d'entrée standard et envoie ces données triées par le canal de sortie standard.

Tr permet de convertir des caractères. La commande tr n'est pas en mesure de lire dans un fichier. Elle lit ses données par le canal d'entée standard et retourne le résultat par le canal de sortie standard.

grep [options] modèle-de-critère [fichier(s)]

Cette commande lit ses données à partir du canal d'entée standard ou directement dans un fichier, si son nom apparaît comme paramètre la commande cherche, dans les données les lignes contenant le critère de sélection (modèle de critère).

Le modèle de critère peut contenir des caractères spéciaux et ainsi former des expressions régulières. Les options influent sur la méthode de travail de la recherche

Exemple d'options :

-c : la commande n'affiche que le nombre de lignes

-i : lors de la comparaison la commande grep ne tient pas en compte de la différence de casse.

-n : cette option numérote les lignes trouvées.

Les Caractères spéciaux :

[...] : Indication d'une plage, pour une série de caractères autorisés à cet emplacement.

[^...] : Négation de la plage indiquée, à cet emplacement peuvent survenir tous les caractères qui ne sont pas stipulés entre crochets.

. : N'importe quel caractère, y compris l'espace.

* : caractère de répétition, agit sur le caractère ou le caractère spécial précédent.

^ : Symbolise le début de la ligne.

S : Symbolise la fin de la ligne.

\{...\} : caractère de répétition.

cut : permet de couper une partie de ligne dans l'ensemble du fichier

Options ;

-dcaractère : indique le séparateur à employer

-fchamp : indique le(s) nom(s) des champs (-f2 ;fl-3 ;fl,4,5 ;fl-3, 5)

shift [nbre]

Par la commande shift, la valeur du 1er paramètre de position est supprimée, et la valeur du second paramètre passe dans le 1er. La valeur du 3cmc vient prendre place dans le second et ainsi de suite. Le nombre de paramètre de position est réduit d'une unité. Si la commande shift est dotée d'un nombre, les paramètres de position sont décalés d'un nombre spécifié.

tail [option nbre] [fiehier(s)]

Affichage des nbre dernières lignes du fichier ou d'un ensemble de données lues par le canal d'entrée standard. Cette commande renvoie par défaut les 10 dernières lignes du fichier**.**

Exemple :

tail -6 f1 (donne les 6 dernières lignes de f1)

head [option] [fichiers]

Affichage des nbre premières lignes du fichier ou d'un ensemble de données lues par le canal d'entrée standard.

Cette commande renvoie par défaut les 10 premières lignes du fichier.

Exemple :

head -5 Fl (donne les 5 premières lignes de Fl)

wc [-option (-l,-c,-w)][fichier(s)] : cette commande permet de compter le nombre de ligne, le nombre des caractères et le nombre des mots.

Let argument(s) : cette commande exploite les arguments comme une expression arithmétique (le signe $ n'est pas obligatoire).

Exemple :

Let "a=a*3 " ou bien let a=a/3

Les opérateurs sont : + ; - ; * ; / ; % ; > ; < ; >= ; <= ; ==;!=; && ; | |

expr argument operateur argument : cette commande exploite l'expression qui l'accompagne sous forme de trois paramètres. Elle est utilisée avec des expressions numériques ou des opérations sur les chaînes de caractères.

TP5 : Programmation Shell (2)

Préparation: Placez-vous sous votre répertoire de connexion et créez un nouveau répertoire nommé TP5 dans lequel vous travaillerez pour répondre aux exercices suivants.

Exercice 1 :

1. Écrire un script Shell "**Inverse**" qui permet d'afficher l'inverse d'un mot donné en argument.

2. Utiliser ce dernier script pour élaborer un nouveau script "**Pal**" permettant de vérifier si le mot passé en argument est palindrome ou non.

Exercice 2 :

En utilisant la commande **find** et la boucle **while,** écrire un script Shell "CherchFich" qui permet qui permet de réaliser un ensemble d'opérations sur un répertoire passé en argument (par défaut le répertoire courant). Ces opérations sont les suivantes:

1. L'affichage du menu suivant :

********** Menu **********

<1> Affichage de la liste des fichiers spéciaux

<2> Affichage des répertoires de taille supérieure à 512 Octets

<3> Affichage des fichiers qui ont été accédé depuis 8 jours

<4> Affichage des fichiers qui ont été créés depuis 3 jours

<5> Affichage des fichiers qui n'ont pas été modifiés depuis 5 jours

<6> Affichage des contenus des fichiers ordinaires qui ont les droits d'accès 755

<7> Quitter

****

2. La demande à l'utilisateur de choisir une option de ce menu.

3. L'exécution la commande correspondante au choix.

N.B : le script doit répéter ces opérations tant que l'utilisateur n'a pas demandé de quitter.

Exercice 3 :

En utilisant la structure **select**, écrire un script "**Formattage**" qui permet de réaliser le menu suivant :

********** Menu**********

<1> Afficher un fichier donné à l'envers

<2> Numéroter les lignes d'un fichier donné

<3> Afficher le contenu d'un fichier donné sous une autre forme indiquée par l'utilisateur

<4> Réduire les espaces entre les mots d'un fichier donné à un espace

<5> Convertir les tabulations d'un fichier donné en espaces

<6> Fusionner les lignes de deux fichiers donnés

<7> Découper un fichier donné selon un nombre de lignes indiqué par l'utilisateur

<8> Quitter

Complément du TP5

find rép_de_départ [critères] [option de cdes]

Cette commande sert à rechercher des fichiers. Tous les répertoires spécifiés (paramètre rép_de_départ) et leurs sous-répertoires sont parcourus de manière récursive.

Critère :

-name fichier : recherche par nom de fichier (caractères spéciaux : * , ? , […])
-type type : recherche par type de fichier (f : fichier ordinaire, d : répertoire, c : fichier périphérique de type caractère, b : fichier périphérique de type bloc, s : socket).
-user nom : recherche par propriétaire.
-group nom : recherche par groupe.
-size nombre : recherche par taille (pour les octets, il faut ajouter un c).
-empty peut être utilisé en remplacement de -size 0.
-atime jour : recherche par date de dernier accès.
-mtime jour : recherche par date de dernière modification.
-ctime jour : recherche par date de création.
-perm droits : recherche par droits d'accès.
-links nombre : recherche par nombre de liens.
-inum permet une recherche par numéro d'inode. Elle est utile dans le cas d'une recherche de tous les liens portant un même numéro d'inode.

Options de commande :

-print : affiche le résultat.
-exec : introduit une commande si un fichier est trouvé.
-ok : même chose que exec mais avec une demande de confirmation.
-ls : des informations détaillées sur les fichiers trouvés correspondant au critère au lieu du simple nom de fichier.

Remarque :

- Pour toutes les options en dehors de –name, -type, -user, -group et –perm, les noms d'options sont complétés par des valeurs. Ces nombres peuvent aussi être précédés des signes + (supérieur à) ou – (inférieur à).
- La commande placée derrière –exec doit se terminer par ;. Comme les points virgules sont des caractères spéciaux, ils doivent être masqués par un backslash.
- Si, dans la commande placée derrière –exec, vous voulez accéder au fichier qui vient d'être trouvé, vous utiliserez l'abréviation {} (deux accolades l'une après l'autre).
- Si plusieurs options sont spécifiées, elles sont liées par –a ou –o. On utilise aussi ! pour la négation. Ces options liées doivent être placées entre parenthèses. Comme ces parenthèses sont des caractères spéciaux, il faut les masquer par un backslash.

tac: affiche le contenu inversé d'un fichier.
nl : numérote les lignes d'un fichier.
od : afficher le contenu d'un fichier en octal ou sous d'autres formes (decimal, hexadecimal, etc.)
option : -t type
exemples : od –t o file1
od –t x file1
fmt : formater les paragraphes dans un fichier.
exemple : fmt –w myfile.txt > myfile80wide.txt

paste : regrouper les lignes de différents fichiers.

paste file1 file2
paste –d'@' file1 file2
paste –s file1 file2

split : découper un fichier en différentes partie.

split –2 file1 splitout_ : créer trois fichiers splitout_aa, splitout_ab, splitout_ac.

split –b 1.4m grosfichier petitfichier_ : découper un fichier en plusieurs de taille maximale d'une disquette.

expand : convertir les tabulations d'un fichier en espaces.

unexpand : fait le processus inverse de expand.

TP6 : Processus

Utilisation des commandes du shell :

ps donne la liste des processus courants

kill permet de tuer un ou plusieurs processus

Exercice 1 :

1. Testez la commande ps.

2. L'option -l de ps permet d'afficher plus de détails sur les processus. Testez cette option et repérez les caractéristiques précédemment évoquées. Détaillez, en vous aidant, au besoin du manuel, la signification des colonnes UID, TT, VSZ, TIME, PRI, NI.

3. Comment peut-on afficher la liste de tous les processus du système ? Repérer le processus init.

4. Comment peut-on regrouper les processus par propriétaire ?

5. Testez la commande pstree, que permet-elle ?

6. Testez la commande top, que permet-elle ?

7. Quelle est la commande qui permet de changer la priorité d'un processus ?

Exercice 2 :

La commande kill permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, elle ne sert pas *seulement* à "tuer" un processus. Il existe de nombreux signaux différents dont vous trouverez la signification dans le **man**. Les signaux les plus courants sont SIGTERM et SIGKILL, qui servent à terminer un processus.

1. La liste des signaux que l'on peut envoyer aux processus s'obtient grâce à l'option -l de kill. Quels sont les numéros correspondant aux signaux SIGTERM, SIGKILL, SIGSTOP, SIGCONT et SIGHUP ?

La syntaxe d'envoi d'un signal est : **kill -*signal pid*** où *signal* est un numéro ou un nom de signal (le signal par défaut est SIGTERM).

2. Quelle est la différence entre ces 5 signaux ?

3. Lancez un second terminal, puis testez les signaux SIGTERM et SIGKILL sur le processus de ce nouveau terminal. Que constatez-vous ? Qu'en déduisez-vous sur le fonctionnement des signaux SIGTERM et SIGKILL ?

4. Repérez parmi les processus actifs un processus dont vous n'êtes pas propriétaire, et tentez de le stopper à l'aide du signal SIGSTOP. Que se passe-t-il et qu'en déduisez-vous ?

5. Quelle est la commande qui nous permet d'envoyer un signal à un processus par son nom et non pas par son pid?

6. Quelle est la commande qui nous permet d'ignorer le signal N°1?

Exercice 3 :

Créez un fichier contenant le script suivant :

```
# !/bin/sh
while true
do
        sleep 10
        echo $0 est toujours la
done
```

1. Lancez ce script en arrière-plan (utiliser &). Lancer, ensuite, la commande ls –l en arrière-plan.
2. A l'aide de quelle commande peut-on repérer les processus lancés en arrière-plan ?
3. Basculez l'exécution du script en premier plan à l'aide de la commande fg, puis le rebasculez en arrière-plan à l'aide de la commande bg.
4. Tuez ce processus.

TP7

Questions :

1. Déterminez le type de votre disque dur (IDE ou SCSI).
2. Déterminez la géométrie de votre disque dur (nombre de plateaux, nombre de pistes, nombre de secteurs)
3. Quelle est la commande permettant de mettre votre disque-dur en mode lecture seule ?
4. Quelle est la commande permettant d'activer/désactiver le cache d'un disque-dur ?
5. Donnez deux méthodes pour déterminer la mémoire totale disponible dans le système.
6. Que fait la commande dmesg ?
7. Examinez le contenu des dossiers (ou fichiers) suivants : /var/log/dmesg, /proc, /proc/dma, /proc/ioports et /proc/interrupts.
8. Quels sont les périphériques qui partagent la même ligne d'interruptions ?
9. Affichez la liste matérielle et l'enregistrez dans un fichier HTML.
10. Affichez la liste des périphériques USB.
11. Affichez la liste des périphériques PCI.
12. Utilisez l'utilitaire lspci avec les options correctes pour obtenir le dessin de l'architecture PCI de votre système.
13. Utilisez l'utilitaire lspci avec les options correctes pour afficher les ressources PCI allouées par le système et celles allouées par le BIOS.
14. Quelles sont les options de lspci permettant d'établir la liste de périphériques PCI Intel ?
15. Combien y a-t-il de bus PCI ?
16. Y-a-t-il des bridges PCI ou ISA ?
17. Combien y a-t-il de bridges ISA ?
18. Y-a-t-il des cartes ISA-PNP sur votre machine ?
1. Consultez la page man de setpci et :
   (a) Déterminez le rôle de cet utilitaire
   (b) Montrez un exemple d'accès à un registre.

TP 8

Exercice 1 : Les droits spéciaux (SUID, SGID, Sticky bit)

1) Comment connaitre les commandes dont la permission est SUID ?
2) A quoi sert la permission t sur un répertoire ? et sur un fichier exécutable ?
3) Que se passe-t-il si le bit sgid est mis sur un répertoire ?
4) Donner l'équivalent de ces droits en octal :
5) Comment rendre un fichier en ajout seulement avec la commande chattr ? Est-ce que cela est valable aussi bien pour des systèmes de fichiers de la famille ext2 que ext3 ?

Exercice 2 : Disques durs et partitionnement

1) Citer les différents types de partions.
2) Combien de partition utilise au minimum Linux ? Essayer de les repérer à l'aide de la commande fdisk -l.
3) Quel est le plan de partitionnement de votre système et combien de disques avez-vous ?
4) Quelle est la commande permettant d'afficher la liste des systèmes de fichiers actifs ?
5) Quelle est la commande qui vous permet de visualiser le contenu de votre flashdisk ?
6) La partition de swap a-t-elle besoin d'un système de fichiers.
7) Quelle est la taille de votre espace de swap ?
8) Quelle est l'option de la commande swapon permettant d'afficher les informations sur la partition de swap ?
9) Comment peut-on obtenir des informations sur l'utilisation de la mémoire virtuelle ?
10) Donner l'instruction générale permettant la création d'un système de fichiers ?
11) Quelle est la commande permettant la vérification de toutes les partitions au démarrage du système ?
12) Donner l'instruction générale permettant la vérification du système de fichiers.

13) Quelle est la commande qui permet de connaître le taux d'utilisation de toutes les partitions montées du système ?

14) Quelle est la commande qui permet de connaître l'espace occupé par une arborescence ?

15) Comment est représenté le périphérique IDE esclave du deuxième contrôleur ?

Exercice 3 : Les quotas

1) Quelle est la commande qui permet de changer le propriétaire d'un fichier ou d'un répertoire ?

2) Quelle est la commande qui permet de changer le groupe d'un fichier ou d'un répertoire ?

3) Quelle est l'utilité des quotas ? La création des quotas fonctionne-t-elle avec tous les systèmes de fichiers ?

4) Quelle est la commande qui permet d'afficher un rapport des quotas d'une partition donnée ?

5) Quelle est la commande qui permet de copier une configuration de quota d'un utilisateur à un autre.

Exercice 4 : Installation des paquetages

1) Quelle est la commande qui permet de redémarrer le système ?

2) Citer les deux méthodes d'installation d'application utilisées par le système d'exploitation Linux.

3) Quelle est la structure du nom d'un fichier rpm ?

4) Quelles sont les options de la commande rpm qui permettent d'installer un paquet, le mettre à jour et le désinstaller ?

5) Que fait la commande dpkg –r ? Quel est son équivalent avec la commande apt-get ?

6) Quel est le fichier qui définit les sources des paquetages Debian à installer ?

7) Y-a-t-il un moyen pour basculer d'un paquet en format rpm à un paquet en format dpkg et vice-versa ?

8) Quelle est la différence entre les bibliothèques statiques et dynamiques ?

Complément du TP8

Aux droits de base (r, w et x) s'ajoutent des droits supplémentaires un peu particuliers. Ce sont:

- le "suid" (ou "setuid") qui concerne les programmes:
    - un programme lancé avec ce droit "suid" sera exécuté avec les droits du propriétaire du programme et non les droits de l'utilisateur qui l'a lancé.
- le "sgid" (ou "setgid") qui concerne les programmes et les répertoires:
    - un programme lancé avec ce droit "sgid" sera exécuté avec les droits du groupe du programme et non les droits du groupe de l'utilisateur qui l'a lancé.
    - un fichier créé dans un répertoire ayant le droit sgid aura pour groupe le groupe du répertoire et pas celui de l'utilisateur qui a créé le fichier.
- le "sticky bit" qui concerne surtout les répertoires:
    - dans un répertoire avec ce droit "sticky bit", seuls les propriétaires des fichiers pourront les effacer.

Dans la présentation des droits en texte, ces droits particuliers seront représentés comme suit:

- pour le suid: un "s" à la place du "x" du propriétaire comme dans "rwsr-xr-x".
- pour le sgid: un "s" à la place du "x" du groupe comme dans "rwxr-sr-x".
- pour le stiky bit: un "t" à la place du dernier "x" (celui du "reste du monde") comme dans "drwxrwxrwt".

NB: s'il n'y avait pas de droit d'exécution "x" avant d'appliquer ces droits, les lettres "s" et "t" seront mises en majuscule ("S" et "T").

Dans la présentation en nombre octal, ces droits correspondent à un 4ème chiffre octal situé à gauche, avec:

- suid=4
- sgid=2
- sticky bit=1