UNIVERISTE MOHAMMED PREMIER
Faculté des Sciences d'Oujda
Oujda - Maroc

Polycopié
Cours Système d'Exploitation I

Programme : Filière – SMI – S3

Pr. El Mostafa DAOUDI

Département de Mathématiques et d'Informatique

Année universitaire 2014/2015

Contenu pages

Chapitre 1 : Introduction aux Systèmes d'Exploitation

Chapitre 2 : Les commandes de base du Système Unix

Chapitre 3 : La programmation Shell

Chapitre 1

Introduction aux Systèmes d'Exploitation

I. Notions générales sur les Systèmes d'exploitations

N.B. Le cours complet sur les systèmes d'exploitation est programmé en semestre S4.

1. Définitions

- L'ordinateur est un ensemble de ressources matérielles (hardware) exemple processeur, mémoire, disque dur, …
- Un Système d'Exploitation (SE), Operating System en anglais (OS), est un ensemble de programmes (logiciels) qui permettent d'assurer la bonne gestion de l'ordinateur et de ses périphériques.

  ⇒ Il sert d'interface entre les ressources matérielles d'un ordinateur et les applications informatiques de l'utilisateur (software). Il cache les spécificités complexes du matériel.

  ⇒ Il est chargé du bon fonctionnement d'un ordinateur en fournissant à l'utilisateur un environnement plus facile à utiliser que le matériel:

  - Il cache les limitations physiques (nombre de processeurs, taille mémoire).
  - Il facilite le partage et l'utilisation des ressources physiques entre les différents programmes (plusieurs programmes peuvent être exécutés simultanément).

Exemples de systèmes d'exploitation :

Windows, Unix, Linux

2. Rôle d'un système d'exploitation

Le système d'exploitation contrôle et coordonne l'utilisation du matériel: Il met à la disposition des utilisateurs, les ressources matérielles de l'ordinateur :

Gestion des ressources matérielles: le système gère de manière équitable et efficace les ressources matérielles (mémoire, processeur, périphériques, …).

- Gestion du processeur: le système d'exploitation gère l'allocation du processeur entre les différents programmes. Pour l'utilisateur, les différents programmes fonctionnent parallèlement.
- Gestion de la mémoire: le système d'exploitation gère l'espace mémoire alloué à chaque application et à chaque utilisateur. Il la partage entre tous les programmes. En cas d'insuffisance de mémoire physique, le système d'exploitation peut créer une zone mémoire sur le disque dur, appelée «mémoire virtuelle», qui permet d'exécuter des applications nécessitant plus de mémoire qu'il n'y a de mémoire vive disponible sur le système.

Sécurité / Accès aux données : Accès aux périphériques: écran, imprimante, disque dur, réseau. Le système d'exploitation s'assure que les programmes puissent les utiliser de façon standard.

3. Système d'exploitation UNIX

UNIX est un système d'exploitation multi-tâches et multi-utilisateurs. Il est:

Ouvert, c'est-à-dire il n'y a pas de code propriétaire (seules certaines implémentations sont propriétaires).

Portable, c'est-à-dire le code est indépendant de l'architecture (très peu de codes qui dépendent de l'architecture matériel de l'ordinateur).

Disponible sur différentes plateformes. La grande majorité des serveurs sur Internet fonctionnent sous UNIX.

Aujourd'hui, UNIX est très utilisé en informatique scientifique, et pour les serveurs réseaux.

Caractéristiques du système UNIX

- Unix est un système d'exploitation multi-tâches (multithreaded en anglais): plusieurs processus (process en anglais), également appelées « tâches », peuvent être exécutées simultanément.

  ⇒A chaque instant, le processeur ne traite qu'un seul processus (programme lancé), la gestion des processus est effectuée par le système.
- Unix est un système d'exploitation multi-utilisateurs (multi-user): plusieurs utilisateurs peuvent utiliser le système en même temps (les ressources sont réparties entre les différents utilisateurs). Chaque utilisateur dispose de l'ensemble des ressources du système.

  ⇒ Le système Unix se charge de contrôler et de gérer l'utilisation et l'attribution des

ressources entre les différents utilisateurs.

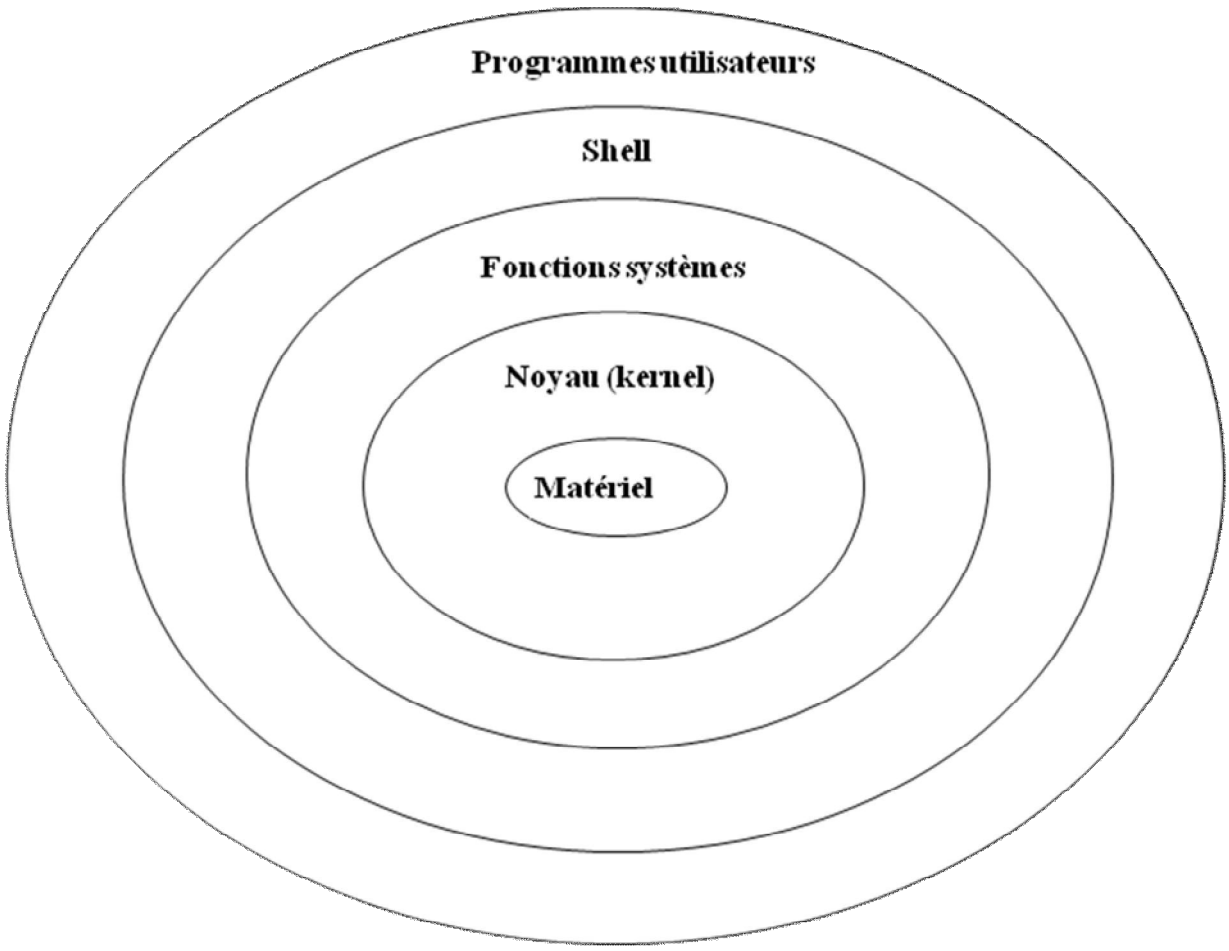
- Unix présente une interface utilisateur interactive et simple à utiliser: le shell. Cette interface fournit des services de haut niveau. Elle intègre un langage de commandes très puissant (scripts shell).
- Sous Unix, du point de vue utilisateur, il n'y a pas de notion de disque physique (partition, disque externe, …) contrairement à MS-DOS, en effet sous Unix, tout est fichier. L'utilisateur ne voit qu'une seule arborescence de fichiers hiérarchiques.
- Les périphériques sont aussi représentés par des fichiers, ce qui rend le système indépendant du matériel et par conséquent assure la portabilité; l'accès aux périphériques est donc identique à l'accès aux fichiers ordinaires.
- La gestion de la mémoire virtuelle : un mécanisme d'échange entre la RAM et le disque dur permet de pallier au manque de RAM.
- Processus réentrants : les processus exécutant le même programme utilisent une seule copie de celui-ci en RAM.

  Exemple: deux utilisateurs qui utilisent l'éditeur « vi », dans ce cas une seule copie de « vi » qui sera chargée en RAM.

4. Architecture du système Unix

Un système informatique sous Unix/Linux est conçu autour d'une architecture en couche:

- **La couche physique (hardware):** c'est la couche la plus interne: ressources matérielles (processeur, mémoires, périphériques,…).
- **Au centre le noyau (en anglais kernel):** le noyau UNIX est chargé en mémoire lors du démarrage de l'ordinateur. Il gère les tâches de base du système à savoir: la gestion de la mémoire, des processus, des fichiers, des entrées-sorties principales, et des fonctionnalités de communication.
- **Fonctions systèmes :** bibliothèque standard d'appels système.
- **L'interpréteur de commandes (le shell en anglais : coquille en français):** c'est la partie la plus externe du système d'exploitation. Son rôle est d'analyser la commande et envoie des appels au noyau en fonction des requêtes des utilisateurs. C'est l'interface utilisateur-Système. C'est le premier langage de commandes développé sur Unix par Steve Bourne.
- **Utilitaires :** éditeurs, compilateurs, gestionnaire de fenêtres et de bureau, etc.



II. Bref historique d'Unix.

- UNIX est créé au Laboratoire BELL (AT&T), USA, en 1969. Il est conçu par Ken Thompson et Dennis Ritchie, et inspiré du système Multics (MULTiplexed Information and Computing Service ou service multiplexé d'information et de calcul) créé en 1965 au le MIT (Massachusetts Institute of Technology). Il constitue le premier système d'exploitation multitâche et multiutilisateur.
- Initialement nommé Unics (Uniplexed Information and Computing Service)
- En 1973, le système est réécrit en langage C (langage développé par Dennis Ritchie) ce qui l'a rendu simple à porter sur de nouvelles plateforme ce qui lui a donné un véritable succès.
- Depuis la fin des années 70, deux grandes familles d'UNIX.
  - Une version développée essentiellement par l'université de Berkeley (Californie), et

nommée UNIX BSD (Berkeley Software Distribution).

- Une version nommée UNIX Système V commercialisé par AT&T.

- Projet GNU (1983) : objectif de développer un SE libre.
- Linux (1991) : un noyau UNIX libre développé par Linus Torvald (étudiant à l'université d'Helsinki)

  ⇒ Premier OS complet GNU/Linux libre. Linux est à la base d'une réécriture de Minix (1987: Andrew Tanenbaum, professeur à l'université libre d'Amsterdam a créé le système d'exploitation Minix). La version 1.0 en 1994, qui donne naissance à la distribution d'un système d'exploitation entièrement libre, GNU/Linux.

Linux (fera l'objet dans la suite des TP)

- Système d'exploitation de type UNIX pour PC, initialement créé par Linus Torvalds ensuite un grand nombre de développeurs bénévoles ont participé à son développement.
- Linux est gratuit et les codes sources sont disponibles (on a le droit d'étudier et de modifier le code source).
- Principaux composants:
  - Noyau: coeur du système, fournit aux logiciels une interface pour utiliser le matériel.
  - Interface graphique X et shell (interpréteur de commande).
  - Nombreux utilitaires et programmes disponibles: compilateur GCC, éditeur Emacs, ….
- Disponible sur de nombreux ordinateurs (super-calculateur, PC, PDA).
- Interactif et batch.

III. Connexion et déconnexion

Puisque Unix est un système multi-utilisateurs, alors il comporte les mécanismes d'identification et de protection permettant d'éviter toute interférence entre les différents utilisateurs. On distingue deux types d'utilisateurs: les administrateurs systèmes et les utilisateurs normaux:

- L'administrateur système appelé aussi « root », utilisateur privilégié ou super utilisateur (super user). Il dispose de tous les droits sur la machine et le système Unix. Il s'occupe de l'administration du système, en particulier il crée les comptes des utilisateurs.
- L'utilisateur normal dispose des droits réduits qui sont définis par l'administrateur système.

Unix associe à chaque utilisateur (un compte):

- Un nom d'utilisateur ou nom de connexion (appelé « login »).
- Un mot de passe (password en anglais),
- Un Home Directory (répertoire de l'utilisateur ou répertoire de connexion),
- Un langage de commandes (shell).

Donc à chaque connexion, le système demande aux utilisateurs leur login et leur mot de passe pour pouvoir travailler sur la machine. Si les deux sont valides alors Unix initialise l'environnement et ouvre une session de travail.

1. Connexion

Pour se connecter à la machine et ouvrir une session de travail (pour pouvoir travailler sur la machine) il faut s'identifier. Pour cela, il faut:

- Entrer le nom de connexion après le message «login»

  Login : <on tape ici le nom d'utilisateur>
- Entrer mot de passage après le message «password»

  Password : <on tape ici le mot de passe>

Une fois connecté, l'utilisateur de trouve alors dans son propre répertoire de connexion (home directory) correspondant à son login (home directory).

Remarque: Pour des raisons de sécurité, les caractères du mot de passe sont cachés, et la vérification se fait après avoir tapé le login et le mot de passe. Si le login ou le mot de passe et incorrecte, un message d'erreur est alors affiché: « Invalid login name »

Attention: Unix fait la différence entre les minuscules des MAJUSCULES.

2. Déconnexion

Pour terminer la session de travail, la méthode de déconnexion dépend de l'environnement de travail:

- Dans le cas d'un terminal, la commande de déconnexion est: « exit » ou « ctrl-D (^D) »
- Dans le cas d'environnement graphique, la méthode de déconnexion dépend l'interface graphique.

Remarque importante: Si vous éteignez la machine avant d'utiliser les procédures de déconnexion, vous risquez d'endommager les fichiers sur les disques.

IV. Introduction à la notion de système de fichiers (file system).

1. Notion de fichier et de répertoire

Le fichier est la plus petite entité logique de stockage permanent sur un disque ou d'autres supports physiques. Il peut contenir du texte, des données, programmes images. ou des programmes stockés sur un disque. Les fichiers sont classés dans des répertoires (catalogues). Chaque répertoire peut contenir d'autres sous-répertoires, formant ainsi une organisation arborescente.

2. Nomenclature des fichiers

Le nom d'un fichier sous Unix est une suite de caractères, dont la taille peut aller jusqu'à 255 caractères. La plupart des caractères sont acceptés, y compris l'espace (très déconseillé). Cependant quelques caractères sont à éviter

* & ; ( ) ~ <espace> \ | ` ? - (en début de nom)

L'extension n'indique pas le type de fichier, par exemple, un exécutable n'a pas besoin d'avoir une extension particulière. Un fichier qui a l'extension « .exe », ne veut pas dire que c'est un fichier exécutable. En faite le caractère « . » est considéré comme un caractère qui fait parti du nom du fichier et non pas un caractère de séparation entre le nom du fichier et son type (comme le MSDOS). Par conséquent, il est possible que le nom d'un fichier peut contenir plusieurs caractères « . » par exemple : nom_fichier.c.java.cpp

Exemples de noms de fichiers pouvant poser problèmes :

```
nom_fichier*          // problème avec le caractère *
smi(s3)               // problème avec les parenthèses ()
smi&sma               // problème avec le caractère &
resultat juin 2014.xls // problème avec l'espace
-f                    // problème avec le symbole « - » (moins)
```

3. Les différents types de fichiers

Pour l'utilisateur sous Unix, il n'existe pas la notion de disques physiques (tout est fichier).

L'utilisateur ne voit qu'une seule arborescence formée de répertoire et de fichiers. On distingue:

- **Les fichiers ordinaires :** Ce sont soit des fichiers contenant du texte, soit des exécutables (ou binaires), soit des fichiers de données. Le système n'impose aucun format particulier aux fichiers et il les traite comme des séquences d'octets. Contrairement au système MSDOS, on ne peut pas connaitre, à priori, les types des fichiers. Pour connaitre les types des fichiers on utilise par exemple la commande: « **file** ».
- **Les répertoires (les fichiers répertoires):** c'est un ensemble de fichiers ou d'autres répertoires (sous-répertoires) de manière récursive. Ils permettent une organisation hiérarchique.
- **Les fichiers spéciaux :** Ce sont des fichiers qui servent d'interface pour les divers périphériques (terminaux, disques dur, clavier, …). Les opérations de lecture/écriture sur ces fichiers sont directement dirigées vers le périphérique associé. Les fichiers correspondant aux périphériques sont stockés dans le répertoire « /dev » (devices).
- **Les pipes nommés (voir le cours S6) et les liens symboliques (voir plus loins).**

4. Systèmes de fichiers

4.1 Définition

Un système de fichiers (File System en anglais), appelé aussi **système de gestion de fichiers,** est une structure de donnée qui définit l'organisation d'un disque (ou partition d'un disque). Il offre à l'utilisateur une vision homogène et structurée des données et des ressources : disques, mémoires, périphériques. Sous système Unix, tout est fichier, il n'y a pas de notions de disques, partition de disques, périphériques, ….

Les fichiers sont regroupés dans des répertoires et les répertoires contiennent soit des fichiers, soit d'autres répertoires. Une telle organisation génère une hiérarchie de répertoires et de fichiers organises en arbre:

- La racine est désignée par « / » (slash): « / » désigne est le répertoire racine.
- Les nœuds sont les répertoires non vides
- Les feuilles sont les fichiers ou les répertoires "vides".

Sous Unix plusieurs systèmes de fichiers peuvent être rattachés au système de fichiers principal. Chaque système de fichiers peut correspondre physiquement à :

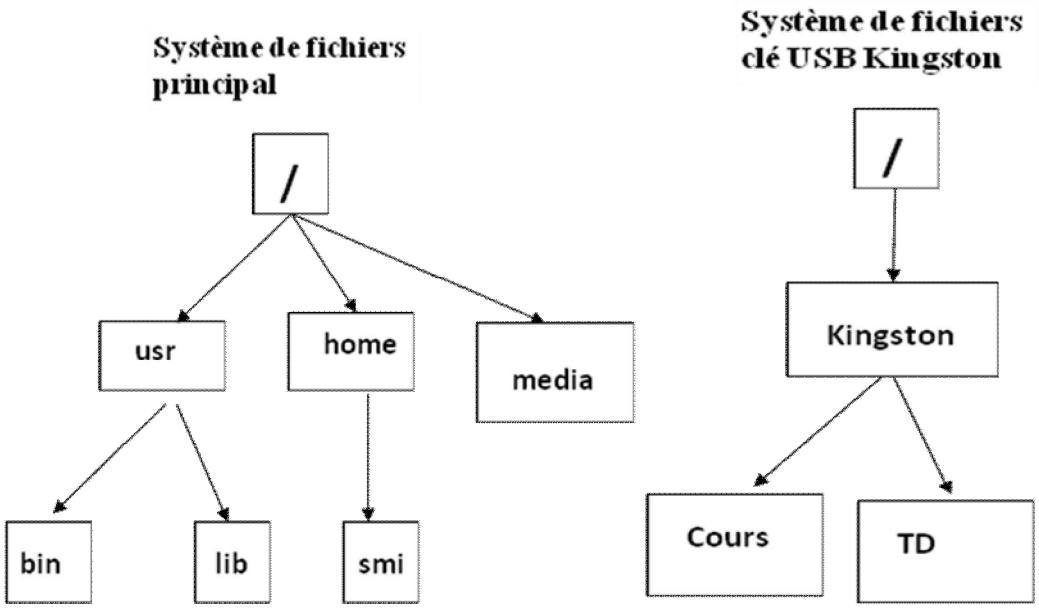
- une partition ou à la totalité d'un disque physique .
- un périphériques (un DVD, un disque externe, …)
- …

Par contre, sous Windows, les partitions, les périphériques, les disques externes, … sont vus comme des lecteurs indépendants (C:, D:, ...).

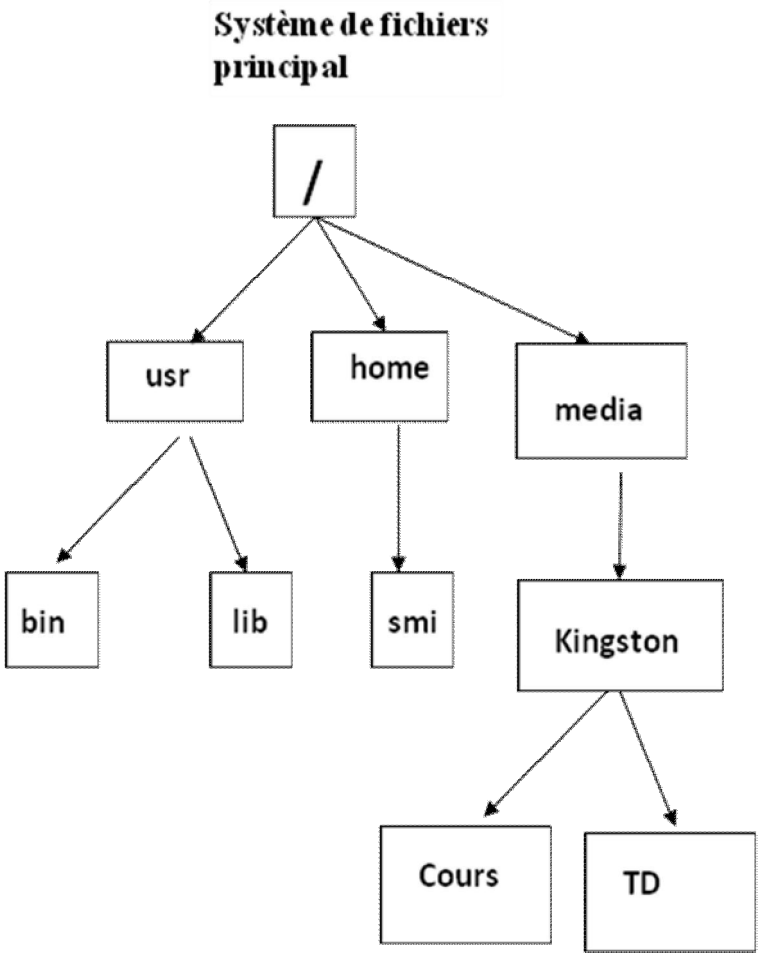
Sou Unix, on a:

- Un seul arbre général.
- Sa racine est désignée par « / ».
- Chaque répertoire peut contenir des fichiers ou des sous-répertoires.
- Un disque logique (partition, disque externe, clé USB, …) est vu comme un sous arbre qui se rattache à l'arbre principal. Le rattachement du sous arbre se fait automatiquement ou par l'utilisateur avec la commande « mount »

Exemple:



Après montage, le système de fichier de la clé USB est rattaché au système de fichiers principal.



4.2 Quelques principaux répertoires Unix

Chaque fichier Unix est placé dans l'arborescence de racine « / » (désigne le répertoire « root ») qui contient les sous-répertoires suivants :

- / bin: utilitaires de bas-niveau (exécutables essentiels pour le système).
- /home : Répertoire pour les utilisateurs. Chaque utilisateur possède son propre répertoire qui a le nom de son login.
- /lib contient des bibliothèques partagées essentielles au système lors du démarrage
- /mnt : contient les points de montage (accès aux autres systèmes de fichiers) des partitions temporaires (cd-rom, disquette, ...)
- /media : Certaines distributions montent les périphériques amovibles à cet endroit.
- /etc: regroupe tous les fichiers de configurations des différents logiciels installés sur la machine ainsi que des fichiers de configuration système utilisés au démarrage de la machine.
- /root: répertoire de l'administrateur root
- /tmp contient les fichiers temporaires

- /var : Fichiers dont le contenu varie

  /var/log : On trouve ici les logs des différents logiciels et serveurs. Cela permet de voir ce qui s'est passé quand quelque chose ne va pas.

  /var/spool : Fichiers en cours de traitement (file d'impression, mails en cours d'envoie...)

  /var/tmp : Fichiers temporaires (voir aussi /tmp).

4.3. Les répertoires « . » et « .. »

Quand on crée un répertoire, le système génère automatiquement deux sous répertoires, du répertoire créé, qui sont:

- le répertoire « . »= représente un lien vers le répertoire créé.
- le répertoire « .. » = représente un lien vers le répertoire père.

Utilités:

- Par exemple pour accéder au répertoire père, il suffit d'écrire « cd.. » au lieu de spécifier le chemin vers le répertoire père.

5. Notion de chemin d'accès (pathname)

Pour accéder à un fichier ou à un répertoire, on doit spécifier son emplacement dans l'arborescence du système de fichier. C'est-à-dire on doit spécifier le chemin d'accès (pathname) qui décrit l'emplacement ou se trouve le fichier dans l'arborescence du système de fichier.

Le chemin d'accès (pathname) est une suite de nom de répertoire séparés par « / » (slach), décrivant l'emplacement où se trouve un fichier ou un répertoire dans l'arborescence du système de fichier. Pour cela, soit on utilise un chemin d'accès absolu ou relatif.

5.1. Chemin absolu:

Un chemin absolu c'est un chemin d'accès dans lequel on spécifie tous les répertoires à partir de la racine, donc le chemin d'accès commence par la racine « / » (root directory).

Exemple: Supposons que « cours » est un répertoire de l'utilisateur « etudiant ». Le chemin absolu pour accéder au répertoire « cours » est : /home/etudiant/cours/

Remarque: Le caractère « / » (slach) désigne le répertoire racine et sert aussi de séparateur entre sous-répertoires.

5.2. Chemin relatif:

Un chemin relatif est un chemin d'accès dans lequel on spécifie tous les répertoires à partir du répertoire. Par conséquent, un chemin relatif ne commence pas par « / »

Exemples:

- Pour accéder au sous-répertoire « exemples » du répertoire « cours » à partir du répertoire « etudiant », le chemin relatif est : cours/exemples
- Pour accéder au répertoire « td » (sous répertoire du répertoire « etudiant ») à partir du répertoire « cours » est : ../TD (« .. » désigne le répertoire « etudiant » : répertoire père).
- Pour accéder au fichier « nom_fich » du répertoire courant :
  Soit on donne le nom du fichier, par exemple « ls nom_fich «
  Soit on précède le nom du fichier par « ./ », par exemple « ls ./nom_fich »

5.3. Le caractère tilde (« ~ »)

Le caractère tilde (« ~ ») désigne le répertoire de connexion d'un utilisateur (le home directory):

« ~ » seul désigne le répertoire de connexion de l'utilisateur.

« ~nom » désigne le répertoire de connexion de l'utilisateur « nom ».

Exemple:

~etudiant/cours : désigne le sous-répertoire « cours » de l'utilisateur « etudiant ».

~/.profile: désigne le fichier « .profile » situé dans le répertoire de connexion de l'utilisateur.

6. Notion d'inode

L'inode (Index NODE ou noeud d'index en Français), est une structure de données qui contient les informations fondamentales à un processus pour accéder aux fichiers, par exemple: le propriétaire du fichier, les droits d'accès, …. Elle est créée au même moment qu'un fichier.

Structure d'un inode :

- Le type (fichier ordinaire, spécial, catalogue ,...),
- Le nombre de liens (voir après),
- UID (User Identification): numéro d'utilisateur du propriétaire
- GID (Group Identification) : numéro du groupe propriétaire
- Taille du fichier en octets

- Adresses des blocs de données (qui contiennent le fichier)
- Droits du fichier
- Date du dernier accès
- Date de la dernière modification
- Date de création

Tables d'inodes

Chaque fichier est référencé dans deux tables :

- une table d'inodes, une table par système de fichiers: cette table regroupe tous les inodes fichier, qui sont regroupés l'un après l'autre. A chaque inode, correspond un numéro d'inode (inumber) qui est son rang dans la table des inodes. Ce numéro est unique au périphérique sur lequel le fichier est situé.
- une table catalogue (une par répertoire) : cette table décrit les correspondances entre les noms de fichiers et les numéros d'inodes. La désignation d'un fichier se fait par l'intermédiaire du répertoire dans lequel il est stocké.

Le système identifie un fichier non pas par son nom, mais par son numéro d'inode.

7. Les liens

7.1. Définition des liens

Puisqu'un fichier est identifié par son numéro d'inode et non pas par son nom de fichier, il est possible de donner plusieurs noms à un même fichier grâce à la notion de lien: ceci permet d'accéder au même fichier à différents endroits de l'arborescence.

Avantage:

- Possibilité d'accéder au même fichier depuis des endroits et des noms différents
  ⇒ Une seule copie sur le disque et plusieurs façons d'y accéder.
- Si l'un des fichiers est modifié, la même modification est prise en compte par l'autre fichier.
- Simplifier l'accès à des fichiers dont les noms (chemin) sont difficiles à retenir.

7.2. Types de liens

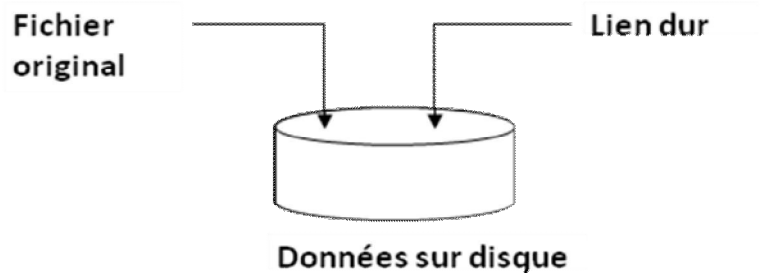
On distingue deux types de liens : les liens symboliques« **symbolic links** » et les liens dur (ou lien physique) « **hard links** ».

Lien physique (hard link)

Un lien dur permet de donner plusieurs noms de fichiers qui ont le même inode (c'est dire des fichiers qui partagent le même contenu). Ceci peut se réaliser en rajoutant de nouveaux noms, dans la table des catalogues, associés au même inode associé au fichier source. Dans ce cas le même fichier physique (inode) est pointé par différents noms de fichiers. Le fichier source et le fichier lien pointent directement sur les données résidant sur le disque (l'information ne réside qu'une seule fois sur le disque mais elle peut être accédée par deux noms de fichiers différents). Les droits du fichier source ne sont pas modifiés.

Remarque:

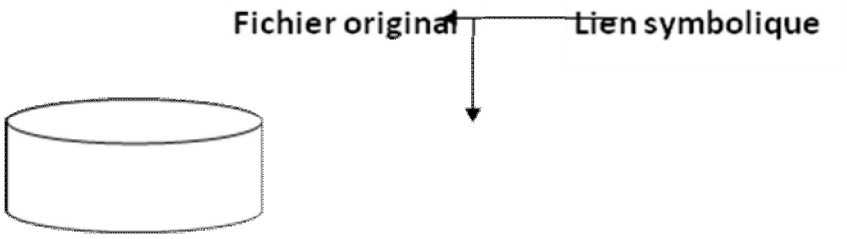
- Si le fichier source est effacé, le contenu n'est pas perdu et est toujours accessible par le fichier lien.
- On ne peut pas faire des liens dur sur des répertoires,
- On ne peut faire des liens durs que dans le même système de fichier. En effet, chaque système de fichiers a sa propre table d'inodes, par conséquent, des fichiers physiques appartenant à des disques distincts peuvent avoir le même inode.



Lien symbolique

Un lien symbolique ne rajoute pas une entrée dans la table catalogue, mais c'est un fichier texte spécial, rajouté dans la table des inodes, qui contient un lien (sorte d'alias) vers un autre fichier ou répertoire. Donc son numéro d'inode est différent du fichier source mais qui pointe sur le fichier source pour permettre d'accéder aux informations sur le disque. Toute opération sur ce fichier (lecture, écriture, …) s'effectue sur le fichier référencé.

Attention: Si le fichier source est effacé, le contenu est perdu et le fichier lien pointe sur quelque chose d'inexistant. L'information est donc perdue.



Remarque:

- Un lien symbolique ne possède pas les limitations du lien dur. Il est possible d'effectuer des liens depuis et vers d'autre système de fichier.
- Possibilité de faire des liens symboliques sur des répertoires.
- La suppression de tous les liens symboliques n'entraîne que la suppression de ces liens, pas du fichier pointé.
- La suppression du fichier pointé n'entraîne pas la suppression des liens symboliques associés. Dans ce cas le lien pointe dans le vide.

V. Les processus

Le système Unix/Linux est multi-tâche car plusieurs programmes peuvent être en cours d'exécution en même temps sur une même machine. Mais à chaque instant, le processeur ne traite qu'un un seul programmes lancés (un seul processus). La gestion des processus est effectuée par le système.

1. **Définitions:** Les processus correspondent à l'exécution de tâches. Plusieurs définitions existent, on peut citer :
   - Un processus est une tâche en train de s'exécuter. Il est doté d'un espace d'adressage (ensemble d'adresses) dans lesquelles le processus peut lire et écrire
   - Un processus est l'activité résultant de l'exécution d'un programme par un processeur. C'est l'image de l'état du processeur et de la mémoire au cours de l'exécution du programme. C'est donc l'état de la machine à un instant « t ».

2. **Quelques données concernant les processus**

- le PID (*Process IDentifier):* chaque processus Unix est identifié par une valeur numérique unique (PID). Le PID du premier processus lancé par le système est 1, c'est le processus « init » père de tous les processus.

Attention: Le PID est lié au processus. Pa exemple, si on lance 10 fois le même programme, on aura 10 processus, et par conséquent 10 PID différents.

- PPID (Prent Process ID): un processus (processus père) peut lancer lui aussi d'autres processus, appelés processus fils. Le processus fils peut identifier son processus père par un numéro désigné par PPID. Tous les processus ont un PPID sauf le processus 0 représente le processus de démarrage du système (qui a créé le processus « init » qui a pour PID 1).
- UID (User IDentifier) et le GID (Group IDentifier) : A chaque processus, on associe l'UID et le GID de l'utilisateur qui a lancé le processus. Les processus enfants héritent de ces informations de son père. Dans certains cas (voir plus) on peut modifier cet état.

3. Les états d'un processus

A un instant donné, un processus peut être dans l'un des états suivants :

actif (en Exécution)

Le processus est en cours d'exécution sur un processeur. Le passage de l'état actif à l'état prêt (interruption) est déclenché par le noyau lorsque la tranche de temps attribué au processus est épuisée.

prêt

Le processus est suspendu provisoirement pour permettre l'exécution d'un autre processus. Le processus peut devenir actif dès que le processeur lui sera attribué par le système.

bloqué (en attente)

Le processus attend un événement extérieur pour pouvoir continuer: Il a besoin d'une ressource pour continuer, une entrée/sortie par exemple, frappe clavier, … . Lorsque la ressource est disponible, il passe à l'état "prêt".

VI. Les entrées/sorties

Lors de l'exécution d'une commande, un processus est créé. Celui-ci va alors ouvrir trois canaux de communication:

- L'entrée standard,
- La sortie standard,
- La sortie d'erreurs standard.

A chacun des trois canaux est affecté un nom de fichier et un numéro :

- Le fichier « stdin »: le processus lit les données en entrée à partir du fichier « stdin ». Il est ouvert avec le numéro logique 0. Par défaut, il est associé associé au clavier.
- Le fichier « stdout »: le processus écrit les sorties qu'il produit dans le fichier « stdout ». Il est ouvert avec le numéro logique 1. Par défaut, il est associé à l'écran.
- Le fichier « stderr » : le processus écrit les messages d'erreur dans le fichier « stderr ». Il est ouvert avec le numéro logique 2. Par défaut, il est associé à l'écran.

Chapitre 2

Les commandes de base du Système Unix

I. Introduction à l'interpréteur de commandes: le shell

1. Définitions : Le shell est un programme (un fichier exécutable) qui a la charge d'analyser et d'exécuter les commandes:

- Il lit et interprète les commandes.
- Il transmet ces commandes au système.
- Il retourne le résultat.

Le shell sert d'interface entre le noyau (le système d'exploitation) et l'utilisateur. Toutes les commandes sont envoyées au noyau à travers le shell;

- soit en ligne de commande,
- soit via une interface graphique.

Remarque : En mode ligne de commandes, on peut utiliser toutes les options des commandes contrairement à l'interface graphique qui n'offre, en général, qu'une partie.

Le reste cours est consacré à l'introduction et l'utilisation des commandes en mode ligne de commande (appelé aussi mode console ou mode interactif).

2. Principe du fonctionnement du shell en mode ligne de commande.

En mode ligne de commande le shell affiche dans un terminal ou dans une console virtuelle, une chaine de caractères appelée « prompt » (ou invite de commande) et attend la saisi d'une commande. Quand on tape une commande suivi de la touche « Entrée , Return: la touche ↵ », le shell exécute cette commande et ensuite réaffiche le prompt et reste en attente sur une nouvelle commande. En générale la convention pour le prompt:

$ ou % pour l'utilisateur normal

pour root (super-utilisateur ou administrateur) dans tous les shells

On peut personnaliser le prompt, par exemple on peut avoir un prompt comme ceci: smi-s3 >

Remarque :

- Pour lancer un terminal depuis l'interface graphique: on utilise le menu du bureau.
- Pour se connecter à une console virtuelle (un écran noir avec une invite de commande), depuis l'interface graphique: on utilise la combinaison des touches « Ctrl+Alt+FN », où N est un chiffre de 1 à 6 (il y a 6 consoles virtuelles désignées par « tty1 » , … « tty6 ».
- Pour revenir au mode graphique depuis une console virtuelle, on utilise la combinaison des touches «Ctrl+ALT+F7 ».

N.B. Le mode graphique est désigné par « tty7 ».

3. Les principaux shells

Il existe plusieurs shells sous Unix :

- **Le bourne shell; sh (/bin/sh):** ancêtre de tous les shells. Il est disponible sur toute plateforme UNIX.
- **Le Bourne again shell; bash (/bin/bash):** version améliorée de « sh ». Disponible dans le domaine public (Fourni le plus souvent avec Linux).
- **Le Korn shell; ksh (/bin/ksh):** Bourne Shell étendu par l'AT&T.
- **Le C shell (C veut dire California); csh (/bin/csh):** Il est développé par BSD. Il offre plus de facilité (rappel des commandes avec les flèches, gérer l'historique des commandes, etc).
- **Le Tenex C shell (TC Shell ou le C shell amélioré); tcsh (/bin/tcsh):** c'est une extension du C Shell (csh).

Le Bourne Shell, le Korn Shell et le Bash Shell sont compatibles entre eux. Le C Shell et le TC Shell sont compatibles entre eux. Par contre, ces deux familles ne sont pas compatibles entre elles. Il est toutefois possible d'exécuter des procédures Bourne Shell alors que le shell de login est le C Shell

- Pour connaitre tous les shells que l'utilisateur peut utiliser, on consulte le fichier « /etc/shells » en exécutant par exemple la commande. **% cat /etc/shells**
- Pour connaitre le shell utiliser, par exemple, on tape la commande **% echo $SHELL**

II. Commandes Unix en mode console

N.B. Sans aucune précisions, toutes les commande dans ce cours sont testées, sous Linux Ubuntu version 10.04, avec l'interpréteur de commandes le « Bourne again shell: bash ».

1. Définition

En général, une commande est un programme. Pour l'exécuter

- On tape son nom, éventuellement suivi d'options et d'arguments.
- A la fin de la saisi, on tape la touche « Entrée » (ou touche « Return » dénotée par ↵), pour valider la saisi.
- Lorsqu'on appui sur la touche « Entrée », le shell exécute la commande.

2. Syntaxe d'une commande:

nom_commande [options] [arguments]

- « nom_commande » est le nom de la commande à exécuter;
- « options »: une ou plusieurs options. Les options permettent de modifier le comportement de la commande.
- « arguments »: les arguments à passer à la commande.

Remarques :

- Les crochets désignent un élément facultatif.
- Chaque mot est séparé des autres par un espace ou une tabulation.
- Une options est composée d'un tiret (-) suivi d'un seul caractère.
- Les options sont séparées par des espaces, par exemple « -a -s -l -i ». Mais il est possible d'accoler les options: Par exemple: -asli désigne les options « -a -s -l -i ».
- Une commande peut avoir plusieurs arguments. Par exemple : % ls /etc /usr
  la commande « ls » a deux arguments /etc et /usr .

Exemples:

% ls

Liste (affiche) tous les fichiers du répertoire courant sauf les fichiers cachés (les fichiers commençant par « . »).

% ls fichA

Si le fichier « fichA » existe, son nom sera affiché. Si le fichier « fichA », n'existe pas alors un message d'erreur est affiché.

% ls -l

L'option -l permet de fournir plus de détails sur les fichiers contenus dans le répertoire courant.

Pour connaitre la localisation de la commande (connaitre le chemin correspondant à une commande) on utilise la commande « which ».

Exemple:

```
% which ls
/usr/bin/ls
```

Le résultat d'exécution montre que la commande « ls » est dans le répertoire « bin » qui est un sous répertoire de « usr » qui est un sous-répéertoire du répertoire « root ».

3. Les Types de commandes

Le shell est un processus, qui s'exécute sur la machine. Quand une commande est saisie:

- Soit c'est le processus shell qui l'exécute (commande interne ou builtin). Une commande interne ne possède pas d'exécutable associé à cette commande. Elle est intégrée au shell lui-même. Lors de l'exécution d'une commande interne, il n'y a pas de création de processus fils pour exécuter cette commande.
- Soit le shell crée un processus fils pour exécuter cette commande (commande externe). Dans ce cas, le processus shell ne pourra pas exécuter une nouvelle commande qu'après la fin d'exécution du processus fils.

Pour identifier les types des commandes, on utilise la commande interne « type ».

Exemple:

```
% type cd
  cd is a shell builtin
  cd est une primitive du shell
```

Le résultat de cette commande montre que la commande « cd » est une commande interne.

```
% type ls
  ls is /bin/ls
```

L'exécution de cette commande affiche le chemin absolu de la commande « ls » qui montre que cette commande est située dans le sous-répertoire « /bin ». Donc la commande « ls » n'est pas une commande interne (n'est pas un builtin).

Liste des commandes internes (builtins)

alias	bg	builtin	bind
cd	chdir	command	echo
eval	exec	exit	export
fc	fg	getopts	hash
jobid	jobs	pwd	read
readonly	set	setvar	shift
trap	type	ulimit	umask
unalias	unset	wait	

Liste des commandes externes

Les commandes externes au shell sont des exécutables (ne sont pas intégrées au shell lui-même) qui sont placées dans des répertoires; par exemple dans « /bin », «/sbin», « /usr/bin », …etc.

/bin/cat	/bin/chmod	/bin/cp	/bin/date
/bin/kill	/bin/ln	/bin/ls	/bin/mkdir
/bin/mv	/bin/ps	/bin/pwd	/bin/rmdir
/bin/sleep	/usr/bin/awk	/usr/bin/basename	/usr/bin/bc
/usr/bin/bg	/usr/bin/chgrp	/usr/bin/cmp	/usr/bin/comm
/usr/bin/cut	/usr/bin/diff	/usr/bin/dirname	/usr/bin/find
/usr/bin/grep	/usr/bin/head	/usr/bin/join	/usr/bin/man
/usr/bin/more	/usr/bin/nohup	/usr/bin/paste	/usr/bin/sed
/usr/bin/sort	/usr/bin/tail	/usr/bin/time	/usr/bin/top
/usr/bin/touch	/usr/bin/uniq	/usr/bin/vi	/usr/bin/w
/usr/bin/wc	/usr/bin/xargs	/usr/sbin/chown	

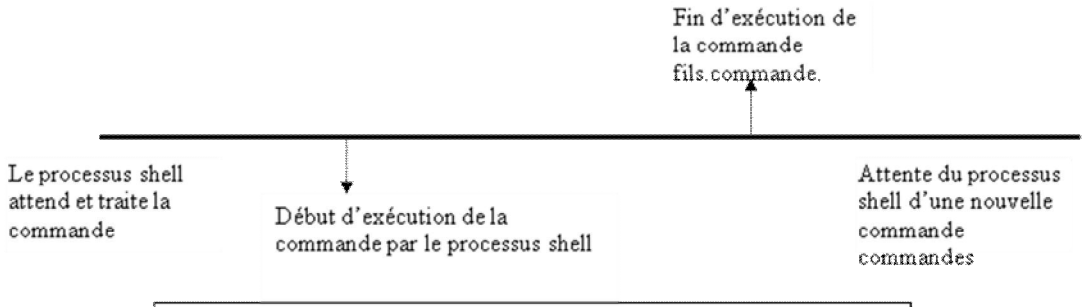


Schéma d'exécution d'une commande interne

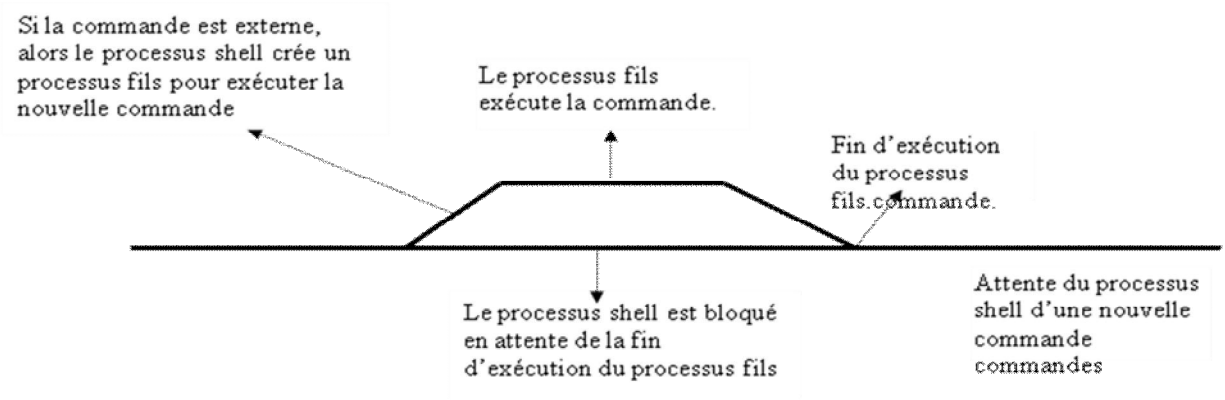


Schéma d'exécution d'une commande externe

4. La documentation Unix

Pour connaitre les différentes options sur les commandes, on peut utiliser la commande «man», qui permet d'accéder au manuel en ligne où toutes les commandes sont documentées.

% man commande.

L'affiche des pages du manuel se fait par page. Pour se déplacer dans le manuel, on utilise

- La touche « ENTRÉE ou RETURN » pour avancer d'une ligne ;
- La touche « ESPACE » pour avancer d'une page ;
- Le caractère « b » pour reculer d'une page ;
- Pour recherche un mot, on utilise le caractère « / » suivi du mot à rechercher ce mot. En suite pour rechercher l'occurrence suivante du même mot, on tape tapez le caractère « n » pour « next » .
- Pour quitter on tape le caractère « q ».

On peut aussi chercher la signification d'un élément, en utilisant la commande « whatis »

Exemple :

% whatis kill

On peut aussi utiliser la commande « help », quand c'est possible, qui donne plus de détail sur la commande, mais moins d'informations que la commande « man ».

Exemple:

% help kill

III. Les commandes liées à l'environnement

1. La commande « lsb_release -a»

Cette commande permet d'afficher les informations sur la distribution Linux utilisée:

```
% lsb_release -a
  Distributor ID: Ubuntu
  Description:    Ubuntu 10.04 LTS
  Release:    10.04
%
```

2. La Commande « hostname »

La commande « hostname » : affiche le nom de la machine.

```
% hostname
  PGR
%
```

3. La commande pwd (Print Working Directory)

La commande « pwd » affiche le chemin d'accès absolu du répertoire de travail courant (le répertoire courant).

Exemple: Supposons que « cours » est un sous répertoire du répertoire de connexion « etudiant »

à partir du répertoire « etudiant »

```
% pwd
  /home/etudiant/
```

à partir du répertoire « cours »

```
%pwd
  /home/etudiant/cours
```

4. La Commande « logname »

La commande « logname » affiche le nom de login de l'utilisateur (le nom de connexion).

Exemple: pour l'utilisateur « etudiant »

```
% logname
  etudiant
%
```

5. La commande « who » :

Cette commande donne la liste des utilisateurs connectés sur la même machine.

Exemple:

```
% who
  etudiant      tty7        2014 -10-20   12:14
  mdaoudi       pts/0       2014 -10-20   14:40
```

L'affichage est de la forme

nom	terminal	date	heure

6. Commande « passwd »:

Cette commande permet à l'utilisateur de changer son mot de passe**:**

```
% passwd
  Old password:
  New password:
  Re-enter password:
```

Pour des raisons de sécurité, il faut taper tout d'abord le mot passe actuel (Old password), pour s'assurer que c'est l'utilisateur lui-même qui veut changer son mot de passe.

N.B. l'administrateur peut définir des règles de sécurité sur les mots de passes, comme par exemple; le nombre minimum de caractères, les types des caractères, … etc.

7. Commande « su »

La commande « su » (Switch User) permet d'ouvrir une session pour un autre utilisateur, par exemple, l'administrateur, peut se connecter en tant que « root », pour des tâches administratives, à partir de la session d'un utilisateur normal.

Remarque: Le mot de passe du nouvel utilisateur sera demandé.

Syntaxe:

su [-] utilisateur

Exemple1 : La commande « su » sans le tiret « - ». Dans ce cas le nouveau utilisateur, après connexion, se trouvera dans le même répertoire de travail avant connexion.

%pwd

/home/etudiant

% su daoudi

passwd: <il faut taper ici le mot de passe de l'utilisateur « daoudi »>

% pwd

/home/etudiant

Exemple2: Commande avec tiret « - » (su - utilisateur). Si le tiret « - » est précisé, le nouveau utilisateur se connecte avec son environnement de travail. Il est conseillé au super utilisateur « root » d'utiliser la commande avec tiret :su -

%pwd

/home/etudiant

% su - daoudi

passwd: <il faut taper ici le mot de passe de l'utilisateur « daoudi »>

% pwd

/home/daoudi

8. Commande « id »: notion d'identité (propriétaire et groupe) sous Unix

L'administrateur du système affecte à chaque utilisateur un identifiant unique (UID : User Identifier»). Il associe aussi à chaque utilisateur, au moins un groupe (groupe principal) : le GID (« Group Identifier»). Les utilisateurs sont définis dans le fichier « /etc/passwd » et les groupes sont définis dans le fichier « /etc/group ». Les commandes

%cat /etc/passwd

et

% cat /etc/group

Permettent d'afficher les contenus de ces fichiers.

Dès qu'un utilisateur est connecté sur une machine Unix/Linux, tous les processus générés,

auront la même identité de l'utilisateur, c'est à dire il seront lancés avec l'UID et le GID de l'utilisateur. La commande « id » permet d'obtenir les informations sur l'UID et le GID

Exemple:

```
% id
  uid=75(mdaoudi) gid=102(fso)
```

9. Autres commandes

- La commande « date »: affiche la date et l'heure.
- La commande « clear » : efface l'écran.
- La commande « exit » : termine le shell (possible aussi CTRL-D, logout)

IV. Commandes liées au file system

1. La commande cd (Change Directory)

La commande « cd » permet de changer le répertoire courant:

Syntaxe

```
cd  [répertoire]
```

- La commande « cd » sans argument, change le répertoire courant par le répertoire de connexion.
- La commande « cd » avec argument, remplace le répertoire courant par le répertoire spécifié. Le répertoire peut être spécifié avec un chemin absolu ou un chemin relatif.

Exemple: Supposons que « cours » et « TD » sont deux sous-répertoires du répertoire « SMI » qui est un sous-répertoire du répertoire de connexion «etudiant».

```
%pwd
  /home/etudiant
% cd  /home/etudiant/SMI/TD          # chemin absolu
%pwd
  /home/etudiant/SMI/TD
% cd ../cours              # chemin relatif
% pwd
  /home/etudiant/SMI/cours
%cd ~daoudi
%pwd
  /home/daoudi
```

2. Les commandes « mkdir » et « rmdir »

2.1. La commande « mkdir » (make directory)

La commande « mkdir » permet de créer des répertoires.

Syntaxe :

mkdir [-p] [repertoire]

- Les arguments de « mkdir » sont les noms des répertoires à créer.
- Si le chemin n'est pas spécifié, le répertoire est créé dans le répertoire courant.
- Si le chemin est spécifié, la commande crée le répertoire dont le nom et le chemin sont spécifiés en argument de la commande. Le chemin peut être relatif ou absolu.

Exemple:

% mkdir projet

crée le répertoire « projet » dans le répertoire courant.

% mkdir ../exam

crée le répertoire « examen » dans le répertoire père du répertoire courant (chemin relatif).

% mkdir /home/etudiant/SMI/tp

crée le répertoire « tp» dans le répertoire « SMI » qui est un sous-répertoire du répertoire de connexion « etudiant » (chemin absolu).

Remarque: Lorsqu'un répertoire est créé, il possède automatiquement deux sous répertoires à savoir « . » et « .. » .

Créer une hiérarchie de répertoires

L'option « -p » (parent) permet de créer un ensemble de sous-répertoire de manière hiérarchique.

Exemple: Créer la hiérarchie de répertoires « SMI/OS/cours » , pour cela :

- Soit on le fait en plusieurs étapes (on utilise 3 fois la commande «mkdir»):

% mkdir SMI

% mkdir SMI/OS

% mkdir SMI/OS/cours

- Soit on le fait à l'aide de l'option « -p » en une seule commande de la manière suivante:

% mkdir -p SMI/OS/cours

Remarque: avec l'option « -p », il n'y a pas de message d'erreurs si le répertoire existe. Dans ce cas la commande ne fait rien.

2.2. La commande « rmdir » (remove directory)

La commande « rmdir » permet de supprimer des répertoires.

Syntaxe:

rmdir [repertoire]

- Les arguments de « rmdir » sont les noms des répertoires existants.
- Si le chemin n'est pas spécifié, le répertoire à supprimer est situé dans le répertoire courant.
- Si le chemin est spécifié, la commande supprime le répertoire dont le nom et le chemin sont spécifiés en argument de la commande. Le chemin peut être relatif ou absolu.

Exemple:

% rmdir tp

Supprime le répertoire « tp » situé dans le répertoire courant.

% rmdir ../exam

Supprime le répertoire « exam» situé dans le répertoire père du répertoire courant.

% rmdir /home/etudiant/SMI/tp

Supprime le répertoire « tp» situé dans le répertoire « SMI » qui est un sous-répertoire du répertoire de connexion «etudiant».

Attention: Les répertoires à supprimer **doivent être vides** (ils ne contiennent que « . » et « .. »).

En cas de plusieurs répertoires à supprimer, la commande « rmdir » supprime les répertoires dans l'ordre dans lesquels ils ont été précisés sur la ligne de commande. Par conséquent, on doit faire attention à l'ordre des arguments.

Exemple:

% rmdir SMI SMI/OS SMI/OS/cours

Affiche un message d'erreur: « echec de suppression « SMI »: le dossier est non vide ».

Il faut respecter l'ordre en commençant par le répertoire le plus interne pour être sûr que le répertoire est vide: % rmdir SMI/OS/cours SMI/OS SMI

Supprimer une hiérarchie de répertoires

L'option « -p » permet de supprimer une hiérarchie de sous-répertoires:

% rmdir -p SMI/OS/cours

3. Les commandes de copies et de déplacements de fichiers

3.1. La commande cp (copy) :

La commande « cp » permet de copier un ou plusieurs fichiers vers un autre fichier ou vers un répertoire.

Syntaxe :

cp [option] fich-source fich_destination

Premier cas: un seul fichier source

- Si « fich_destination » est un répertoire, alors le fichier « fich_source » sera dupliqué dans le répertoire « fich_destination ».

  Attention: si « fich_destination » existe dans le répertoire destination alors il est écrasé et remplacé par le nouveau fichier « fich_source ».

- Si « fich_destination » n'est pas un répertoire, alors la commande « cp » effectue une copie du fichier source « fich_source » vers le fichier destination « fich_destination ».

  Attention: si « fich_destination » existe alors il est écrasé et remplacé par « fich_source ».

Deuxième cas: plusieurs fichiers sources

Si la commande « cp » possède plusieurs fichiers sources alors la destination **doit être un répertoire**. Dans ce cas, la commande « cp », duplique les fichiers sources dans le répertoire spécifié.

Attention: Si l'un des fichiers existe dans le répertoire spécifié alors il sera écrasé et remplacé par le nouveau fichier (le fichier source).

Options de la commande cp

- L'option « -i » : l'utilisateur doit confirmer avant d'écraser un fichier existant.
- L'option « -p » : préservation des permissions, dates d'accès de modification.
- L'option « -r » : récursif. Si le fichier source est un répertoire, alors on copie les fichiers et les sous-répertoires.

Exemple:

- Le fichier destination est un répertoire: copie du fichier «image.jpg» dans le répertoire « SMI »

  % cp image.jpg SMI ou % cp image.jpg SMI/

- Un seul fichier source et le fichier destination n'est pas un répertoire: copie du fichier «

cv.txt » dans le fichier «cv_back.txt»

% cp cv.txt cv_back.txt

- - Plusieurs fichiers sources séparés par « espace », la destination est un répertoire: les fichiers sont copiés dans le répertoire «SMI».

% cp cv.txt cv_back.txt SMI/

3.2. La commande mv (move)

Elle permet de renommer et/ou déplacer un fichier. Sa syntaxe est similaire à la commande « cp ».

Syntaxe :

mv [option] fich-source fich_destination

Premier cas: un seul fichier source

- Si « fich_destination » est un répertoire alors la commande « mv » déplace le fichier « fich_source » dans le répertoire « fich_destination ».

  Attention: si « fich_source » existe dans le répertoire de destination alors il est écrasé et remplacé par le nouveau fichier « fich_source ».
- Si « fich_destination » n'est pas un répertoire, alors la commande « mv » renomme le fichier source « fich_source » en lui donnant le nouveau nom « fich_destination ».

  Attention: si « fich_destination » existe alors il est écrasé et remplacé par « fich_source ».

Deuxième cas: plusieurs fichiers sources:

Si la commande « mv » possède plusieurs fichiers sources alors la destination doit être un répertoire. Dans ce cas, la commande « mv » déplace les fichiers sources dans le répertoire spécifié.

Attention: Si l'un des fichiers existe dans le répertoire spécifié alors il sera écrasé et remplacé par le nouveau fichier (fichier source)

Exemples:

- Un seul fichier source.

% mv cv.txt cv_old.txt

Renomme le fichier source « cv.txt » en lui donnant le nouveau nom « cv_old.txt ».

- Plusieurs fichiers source, la destination doit être un répertoire.

```
% mv  image.jpg   cv_old.txt   cours/    documents/
```

Déplacement des fichiers « image.jpg », « cv_old.txt » et du répertoire «cours» dans le répertoire « documents »

N.B. L'option « -i » : l'utilisateur doit confirmer avant d'écraser un fichier existant.

4. La commande « ls »

La commande « ls » liste le contenu d'un répertoire.

Syntaxe :

ls [options] [arguments]

- La commande « ls » sans arguments, liste le contenu du répertoire courant sauf les fichiers cachés (les fichiers commençant par un point).
  - Si l'argument de la commande « ls » est un nom de fichier alors elle liste ce fichier s'il existe. Sinon elle affiche une erreur.

Exemple:

```
%ls image.eps
  image.eps
```

- Si l'argument utilisé est un nom de répertoire alors la commande « ls » liste le contenu du répertoire passé en argument.

Exemple:

```
% ls
  tp1.c   cv.txt  SMI/
% ls SMI
  figure1.eps  test.c
%cd SMI
%ls
  figure1.eps  test.c
```

Les options les plus utilisées de la commande « ls ».

- L'option « -l »: liste les fichiers avec des informations supplémentaires sur chaque fichier (le type de fichier, les protections, le nombre de liens , le propriétaire, le groupe, la taille en octets, la date de la dernière modification).
- L'option « -a »: affiche tous les fichiers y compris les fichiers cachés (fichiers dont le nom

commence par « . » .

- L'option « -i » : affiche les numéros des inodes des fichiers.
- L'option « -s » : affiche la taille en Ko de chaque fichier.
- L'option « -F »: ajoute un « / » après le nom de chaque répertoire, un «*» après chaque fichier possédant le droit d'exécution et un « @ » après chaque fichier lien.
- L'option « -R »: liste les fichiers et les répertoires de façon récursive
- L'option « -d »: si le paramètre est un nom de répertoire, il vérifie s'il existe et ne descend pas dans un répertoire.

Exemples: Supposons que le répertoire courant contient le sous-répertoire « rep », les fichiers: « fich1 », « fich2 » et «fich3». Supposons que le sous répertoire « rep » contient les fichier «fich » et « fich4 ».

```
% ls
  rep  fich1 fich2 fich3
% ls -a
  . .. .profile rep  fich1  fich2  fich3
% ls -F
  rep/  fich1  fich2*  fich3@
% ls -R
  rep fich1 fich2 fich3
  ./rep:
          fich4 fich5
% ls -d rep
  rep
```

5. Les attributs d'un fichier

A chaque fichier sont assignés les attributs suivants:

- le type,
- le masque de protection,
- le nombre de liens avec d'autres fichiers,
- le propriétaire,
- le groupe,
- la taille du fichier,
- la date de création et/ou de la dernière modification.

Le type:

Dans le tableau suivant, la liste des quelques types avec leurs codes:

Type	Code
ordinaire	-
répertoire	d
lien symbolique	l
pipe nommé	p

Propriétaire et groupe:

A chaque fichier sont associés un UID et un GID définissant son propriétaire et son groupe d'appartenance.

Masque de protection:

Sous Unix, on distingue trois modes d'accès (permissions) :

- L'accès en lecture, désigné par : r (Readable)
- l'accès en écriture, désigné par : w (Writable)
- l'accès en exécution, désigné par x: (Executable)
- Aucun droit d'accès, désigné par –

La signification de chaque mode d'accès dépend du type de fichier (ordinaire ou répertoire).

Pour les Fichiers ordinaires

Droit de Lecture:

L'utilisateur est autorisé à lire le contenu du fichier (le visualiser), le copier, le compiler. Sans ce droit, l'utilisateur ne peut pas, par exemple copier ce fichier ou le compiler.

Droit d'Ecriture:

L'utilisateur peut modifier (écrire dans) le fichier.

Attention: pour pouvoir modifier le fichier il faut l'ouvrir, donc le droit d'écriture seul (sans le droit de lecture) ne permet pas de modifier le fichier.

Droit d'Exécution:

Si le fichier est exécutable (binaire, script shell, …), l'utilisateur peut l'exécuter le fichier.

Pour les répertoires (catalogue)

Droits de Lecture:

L'utilisateur est autorisé à lister le contenu du répertoire (visualiser le contenu du répertoire), par exemple faire exécuter la commande « ls ».

Attention: ce droit ne permet pas d'entrer dans le répertoire (on ne peut pas faire un « cd »).

Droit d'Ecriture:

L'utilisateur est autorisé à créer, supprimer ou renommer des fichiers du répertoire.

Attention: Pour pouvoir modifier le répertoire il faut l'ouvrir, donc le droit d'écriture seul (sans le droit de lecture) ne permet pas de modifier le répertoire.

Droit d'Exécution:

L'utilisateur est autorisé à accéder au répertoire (faire la commande cd). Avec uniquement ce droit les fichiers et répertoires inclus dans celui-ci peuvent être accédés (par exemple lister leur contenus) mais il faut alors obligatoirement connaître leur nom.

Les différents utilisateurs du fichier

Pour chaque fichier, on distingue trois types d'utilisateurs:

- Le propriétaire du fichier: c'est l'utilisateur qui possède le fichier.
- Les utilisateurs du même groupe que le propriétaire du fichier: le groupe auquel appartient le fichier.
- Les autres utilisateurs ayant accès au système: Les autres utilisateurs ayant accès au système.

La commande « ls –l » :

La commande « ls –l » permet d'afficher les attributs des fichiers.

Exemple:

%ls -l

drw-r-xr-x 2 etudiant SMI 4096 2012-04-23 09:14 cours

-rwxr-xr-- 1 etudiant SMI 16491 2014-10-15 10:25 tp1.java

Signification de la première ligne:

drw-r-xr-x 2 etudiant SMI 4096 2012-04-23 09:14 cours

- Le nom du fichier est « cours ».
- type: d (premier caractère) ceci montre que « cours » est un répertoire.

- Masque de protection « rw-r-xr-x »: désigne les droits d'accès pour chaque utilisateur. Ils sont regroupés par bloc de 3 caractères.
  - Le premier bloc de trois caractères « rw- » désigne les droits d'accès pour le propriétaire: le propriétaire peut lire et modifier mais ne peut pas exécuter le fichier « cours ».
  - Le deuxième bloc de trois caractères « r-x » désigne les droits d'accès pour le groupe: les membre du groupe peuvent lire et exécuter mais n'ont pas le droit d'écriture.
  - Le troisième bloc de trois caractères « r-x » désigne les droits d'accès pour les autres utilisateurs: les autres utilisateurs peuvent lire et exécuter mais n'ont pas le droit d'écriture.
- Nombre de liens 2: signifie que le répertoire possède un lien dur (c'est par défaut).
- Propriétaire « etudiant »: désigne le propriétaire du répertoire « cours ».
- Groupe « SMI »: le nom du groupe.
- La taille est de 4096 octets (donné par défaut pour les répertoires).
- La dernière modification sur le répertoire est le 23 Avril 2012.

Signification de la deuxième ligne:

-rwxr-xr-- 1 etudiant SMI 16491 2014-10-15 10:25 tp1.java

- le nom du fichier est « tp1.java »
- type du fichier: « - » (le premier caractère) ceci montre que « image.eps» est un fichier ordinaire.
- Masque de protection « rwxr-xr-- » : désigne les droits d'accès pour le fichier « tp1.java ». Le premier bloc de 3 caractères « rwx » pour le propriétaire, le deuxième bloc de 3 caractères « r-x » pour le groupe et le troisième bloc de 3 caractères « r-- » pour les autres utilisateurs:
- Nombre de liens 1: signifie que le fichier ne possède pas de lien.
- Propriétaire « etudiant »: désigne le propriétaire du fichier «tp1.java »
- Groupe « SMI »: le nom du groupe.
- La taille du fichier « tp1.java » est 16491 octets.
- La dernière modification faite sur le fichier: le 15 Octobre 2014.

6. Effacement d'un fichier ou un répertoire - Commande rm

La commande « rm »(remove) permet de supprimer des fichier ou des répertoires, à condition que les permissions le permettent.

Syntaxe :

rm [options] [argument]

- Si l'argument est un nom de fichier ordinaire alors, il sera supprimé.
- Si l'argument est un répertoire alors il faut rajouter l'option « -r ».
- Si plusieurs arguments sont spécifiés, alors ils seront tous supprimés.

Les options:

- « -r » (récursif): efface le répertoire ainsi que son contenu (fichiers et sous-répertoires).
- « -i » (interactive): demande une confirmation (y ou n) sur chaque fichier à effacer.
- « -f » (force) : supprime le fichier, même s'il est protégé (sans tenir compte des protections du fichier), il ne vérifie que le propriétaire. Vous pouvez donc effacer vos fichiers, même s'ils sont protégés.

Attention : il faut utiliser les options « -r » et « -f » avec précaution.

7. La commande ln

La commande « ln » permet de créer des lien sur un fichier (fichier existant). Si l'un des fichiers, source ou lien, est modifié, les autres le sont aussi.

Syntaxe:

ln [options] fichier_source fichier_lien

- fichier_source: c'est le fichier (ou répertoire) sur lequel on crée un lien.
- fichier_lien: c'est le nom du fichier lien . On peut accéder au fichier source « fichier_source » via le fichier lien « fichier_lien ».

Remarque : Si la destination est un répertoire, alors on crée un lien dans ce répertoire qui a le même nom que le fichier source.

7.1. Les liens symboliques

Pour créer un lien symbolique, on utilise l'option « -s ».

Syntaxe:

ln -s fichier_source fichier_lien

crée un lien symbolique « fichier_lien » contenant la référence du fichier source « fichier_source »
La commande « ls -l » fait apparaître le lien sous la forme

fichier_lien -> fichier_source

Exemple : On crée un lien symbolique « image.eps » sur le fichier «figure.eps ». Le fichier

source est « figure.eps », le fichier lien est « image.eps »

```
% ln -s  figure.eps  image.eps
% ls -l
   -rwxr-xr-x   1 etudiant  SMI   5977  2014-10-16 11:05 figure.eps
   lrwxrwxrwx  1 etudiant  SMI    11     2014-10-16 11:08 image.eps -> figure.eps
% ls -F
   figure.eps   image.eps@
```

Les caractères « @ » (image.eps@) et « -> » (image.eps -> figure.eps) indiquent qu'il s'agit d'un lien symbolique. C'est un fichier spécial de type «l»

Remarques:

- Ce sont les droits du fichier ou du dossier pointé qui seront pris en compte et non les droits du fichier lien.
- Le nombre de liens donné par la commande « ls –l » indique le nombre de liens durs et non le nombre de liens symboliques.
- Un répertoire lien peut être utilisé comme un répertoire normal.
- Les numéros d'inodes des liens symboliques sont différents du numéro d'inode du fichier pointé.

  Exemple: le fichier lien « image.eps » n'a pas le même numéro d'inode que le fichier pointé « figure.eps ».

  ```
  % ls -i
    65484 figure.eps   65496 image.eps
  ```

Attention:

- La suppression de tous les liens symboliques n'entraîne pas la suppression du fichier pointé.
- La suppression du fichier pointé n'entraîne pas la suppression des liens symboliques associés. Mais:
  - Ces liens pointent dans le vide.
  - Si le nombre de liens est égal à 1, on ne peut plus accéder aux données du fichier, ils sont définitivement perdus.

7.2. Les liens physique (lien dur: hard link)

La commande « ln » sans l'option « -s » permet de créer les liens durs.

Syntaxe:

ln fichier_source fichier_lien

Crée le lien physique « fichier_lien » qui contient la référence du fichier source « fichier_source »

Exemple: On crée un lien dur «figure.eps_lien» sur le fichier «figure.eps». Le fichier source est « figure.eps », le fichier lien est «figure.eps_lien»

```
%ln  figure.eps  figure.eps_lien
%ls  -l
  -rwxr-xr-x  2  etudiant  SMI  5977  2014-10-16 11:05   figure.eps
  -rwxr-xr-x  2  etudiant  SMI 5977  2014-10-16 11:16   figure.eps_lien
```

Remarque: La commande « ls -l » liste le lien dur comme un fichier ordinaire. Sur l'exemple, le fichier lien « figure.eps_lien » apparait comme les autres fichiers avec un nombre de lien égal à 2, de même pour le fichier source « figure.eps ».

Pour connaitre, si deux fichiers sont liés par des liens durs, il faut afficher leurs numéros d'inodes, pour cela on exécute la commande « ls -i ».

```
% ls -i
  65484  figure.eps
  65484  figure.eps_lien
```

Les deux fichiers « figure.eps » et « figure.eps_lien » ont le même numéro inode, donc ils sont liés par un lien physique.

Remarque:

- La commande « ls -l », montre que les fichiers « fichier_source » et « fichier_lien » ont chacun un nombre de liens égal à 2. Ceci veut dire que les deux fichiers possèdent des liens durs.
- La commande « ls –i » montre que les deux fichiers ont le même numéro d'inode, donc ils sont liés par un lien physique.
- Le contenu des fichiers (inode) peut être accédé par «fichier_source» ou par

« fichier_lien ».

- Si l'un des fichiers est modifié, l'autre aussi. En faite les deux fichiers accèdent à la même information sur disque.
- Si l'un des fichiers est supprimé, on peut toujours accéder aux données avec l'autre fichier.
- Si l'un des fichiers est supprimé alors le nombre de liens de l'autre fichier sera diminué de 1.
- Si on change les droits d'accès pour l'un, ce changement sera automatiquement effectué sur l'autre fichier.

8. Affichage du contenu d'un fichier

8.1. La commandes « cat »

La commande « cat » permet d'afficher, sur la sortie standard, le contenu des fichiers spécifiés en argument de la commande, l'un après l'autre.

Syntaxe :

cat [options] fichier1 [fichier2,..]

Options:

« -n » : permet d'afficher, en plus, les numéros de lignes des fichiers.

« -b » : identique à « -n » mais les lignes vides ne seront pas numérotées.

Exemple:

Afficher sur la sortie standard le contenu du fichier «/etc/passwd»

% cat /etc/passwd

Remarque:

La commande « cat » permet de faire la concaténation des fichiers en redirigeant la sortie standard à l'aide des opérateurs de redirections « > » ou « >> ».

Exemple: concaténer les fichiers « fiche1 » et « fiche2 » dans « fiche 3 »

%cat fiche1 fiche2 > fiche3

8.2. La commande more

Syntaxe :

more fichiers

Affiche sur la sortie standard, le contenu des fichiers, spécifiés en argument de la commande, l'un après l'autre et page écran par page écran.

- Pour visualiser la page suivante, on tape sur la touche «Space ».
- Pour visualiser la ligne suivante, on tape sur la touche «Entrée» (ou touche « return »)
- Pour terminer la visualisation, on tape sur la touche «q» ou «Q».

8.3. les commandes « head » et « tail ».

Commande « head »

Permet d'afficher sur la sortie standard les débuts des fichiers passés en argument de la commande, l'un après l'autre.

Syntaxe:

head [-q] [-c nbcar] [-n nblignes] [fiche,...]

- L'option « -q » n'affiche pas le nom du fichier.
- L'option « -v » affiche le nom du fichier avant d'en afficher l'entête.
- Par défaut, la commande affiche les 10 premières lignes de chaque fichier passé en argument.

Exemple:

%head fiche1 fiche2

Affiche les 10 premières lignes du fichier « fiche1 » ensuite affiche les 10 premières lignes du fichier « fiche2 »

- L'option « -n »: elle peut s'écrire :

% head -n nb fiche ou **%head -nb fiche**

Affiche les « nb » premières lignes du fichier « fiche »

Exemple:

% head -n 4 fiche ou **% head -4 fiche**

Affiche les 4 premières lignes du fichier « fiche »

- L'option « -c »: elle peut s'écrire :

% head -c nb fiche ou % head -cnb fiche

Permet d'afficher les « nb » premiers caractères du fichier « fiche ».

Exemple:

%head -c 12 fiche ou % head -c12 fiche

Affiche les 12 premiers caractères du fichier « fiche ».

Commande « tail »

Elle permet d'afficher sur la sortie standard les dernières lignes des fichiers passés en argument de la commande fichier.

Syntaxe:

tail [-c nbcar] [-n +nb/-nb] [fiche,...]

- Par défaut, la commande affiche les 10 dernières lignes de chaque fichier passé en argument.

Exemple:

%head fiche1 fiche2

Affiche les10 dernières lignes du fichiers « fiche1 » ensuite affiche les 10 dernières lignes du fichiers « fiche2 ».

- L'option « -n -nb»: elle peut s'écrire :

%tail -n -nb fiche ou % tail -n nb fiche ou %tail -nb fiche

Affiche les « nb » dernières lignes du fichier « fiche »

Exemple:

%tail -n -4 fiche ou %tail -n 4 fiche ou % tail -4 fiche

Affiche les 4 dernières lignes du fichier « fiche »

- L'option « -n +nb» : elle peut s'écrire:

% tail -n +nb fiche

Affiche toutes les lignes du fichier à partir de la ligne « nb »

Exemple:

% tail -n +4 fiche

Affiche le contenu du fichier « fiche » à partir de la 4ème ligne.

9. Les métacaractères du shell

Les métacaractères sont utilisés pour regrouper (générer) les noms des fichiers qui ont des parties communes (qui ont certaines similitudes).

- **Le métacaractère * (jocker) :** Remplace n'importe quelle chaîne de caractères dans le nom

d'un fichier, sauf le carcatère « . » en première position (fichier cachés). Par exemple l'expression «test*» désigne tous les noms de fichiers, relativement à un répertoire, qui commencent par le mot « test ».

Exemples:

- Lister tous les fichiers, du répertoire courant, qui commence avec le caractère « c ».

```
% ls -d  c*
```

- Afficher le contenu de tous les fichiers qui terminent avec « .c » et qui sont localisés dans le répertoire «/home/daoudi/cour»

```
% cat  /home/daoudi/cours/*.c
```

- Lister tous les fichiers dont les noms contiennent « . »

```
% ls -d *.*
```

Attention : «ls *.* » va lister tous les fichiers dont les noms contiennent « . » sauf les fichiers dont les noms commencent par « . ».

Question: pourquoi on rajoute l'option « -d » ???

- **Le métacaractère « ? »:** Remplace (masque) n'importe quel caractère sauf le point (« . ») en première position.

Exemples:

- Lister tous les fichiers et les répertoires qui commencent par la chaine « smi » suivie d'un caractère quelconque et se terminent par « .txt »

```
% ls  smi?.txt
```

- Lister tous les fichiers dont le deuxième caractère est la lettre «b».

```
% ls -d ?b*
```

Attention : «ls ?b* » n'affiche pas les fichiers dont le deuxième caractère est la lettre « b » et qui commencent par « . »

- **Les métacaractères les crochets [ ]**: désigne un seul caractère parmi les caractères qui sont donnés entre crochets (« [ ] »).

N.B . Les caractères sont accolés ou séparés par « , ». Pas d'espaces après le crochet ouvrant et avant le crochet fermant et entre les caractères.

Exemple :

- Lister tous les noms des fichiers et répertoires, du répertoire courant, qui commencent avec la chaine « test.» suivie par la lettre « c » ou par la lettre «j», et qui terminent par n'importe quelles autres chaines de caractères.

```
% ls -d test.[cj]*
```

ou

```
% ls -d test.[c,j]*
```

- **Les métacaractères [-]:** désigne un seul caractère parmi la séquence de caractères définie entre les crochets («[ ]»).

Exemple:

- Lister tous les fichiers et répertoires qui commencent avec une lettre (minuscule ou majuscule) .

```
% ls  [a-zA-Z]*
```

- **Les métacaractères [^] ou [!]:** désigne n'importe quel caractère sauf ceux donnés entre crochets.

Exemple:

- Lister tous les fichiers qui ne commencent pas avec une lettre (minuscule ou majuscule).

```
%ls  -f [!a-zA-Z]*
%ls  -f [^a-zA-Z]*
```

10. Changement de protection

10.1. Commande « chmod »

Dès la création d'un fichier ou d'un répertoire, on lui affecte des droits d'accès par défaut. Ces droits d'accès peuvent être modifiés par le propriétaire du fichier ou l'administrateur du système à l'aide de la commande chmod « **chmod » (CHange Mode).**

Syntaxe:

chmod mode fich

- Changer les droits d'accès du fichier « fich ».
- mode=masque de protections: définit les nouveaux droits d'accès du fichier « fich ». Il dépend de la représentation utilisée : Représentation symbolique ou Représentation numérique.

10.2. Commande chmod: Représentation symbolique

Pour autoriser un accès à un utilisateur:

mode=<utilisateur>+<permissions>

Pour supprimer un accès à un utilisateur:

mode=<utilisateur>-< permissions>

« utilisateur » désigne l'utilisateur auquel on ajoute ou on supprime une permission. Il est codé par:

- « u » (pour user): pour désigner le propriétaire du fichier (user)
- « g » (pour group): pour désigner les utilisateurs du même groupe.
- « o » (pour other) : pour désigner les autres utilisateurs.
- « a » (all): pour désigner tous les utilisateurs (u, g et o).

«permissions » désigne les droits d'accès. Ils sont codés par :

- « r » (pour read): accès en lecture
- « w » (pour write): accès en écriture
- « x » (execute): accès en exécution

Remarque: Lorsqu'on supprime une permission à un fichier, celle-ci sera désignée par le caractère « - »

Exemples:

% ls -l fich

-rw---xr-x 1 etudiant SMI 5977 2012-02-16 11:05 fich

Pour enlever aux autres utilisateurs les droits de lecture et d'exécution.

% chmod o-rx fich

Pour permettre au propriétaire d'exécuter le fichier « fich » (rajouter le droit d'exécution au propriétaire)

% chmod u+x fich

Pour permettre au groupe de lire le fichier « fich » (rajouter le droit de lecture au groupe)

% chmod g+r fich

Les nouveaux droits d'accès du fichier « fich »

% ls -l

-rwxr-x--- 1 etudiant SMI 5977 2012-02-16 11:05 fich

On peut appliquer la méthode « chmod » pour plusieurs utilisateur, dans ce cas on sépare les

modifications par « , » sans espace.

N.B. Pas d'espace avant et après la virgule

Exemple:

```
% ls -l fich
  -rwxr-x---  1 etudiant  SMI  5977  2012-02-16 11:05  fich
```

- Pour enlever au propriétaire le droit de d'écrire et d'exécuter, et permettre au groupe et aux autres utilisateurs le droit d'écrire.

```
% chmod u-wx,go+w fich
% ls -l fich
  -r--rwx-w- 1  etudiant  SMI  5977  2012-02-16 11:05  fich
```

Remarque:

Lorsque le fichier et un répertoire, on peut rajouter l'option « -R » (pour récursif) qui permet d'appliquer les droits sur tous les fichiers et sous-répertoires contenus dans le répertoire, de manière récursive.

On peut utiliser **le symbole « = »** pour exprimer les droits accordés à un utilisateur, les autres droits sont enlevés (il sont mis à « - »). On sépare les droits avec des virgules.

N.B. Pas d'espace avant et après la virgule.

Exemple: Ecrire la commande qui exprime pour le propriétaire les droit de lire, d'écrire et d'exécuter, pour groupe les droits de lire et d'exécuter et pour les autres utilisateurs le droit d'exécuter.

```
% chmod u=rwx,g=rx,o=x  fich
% ls -l fich
  -rwxr-x--x 1 etudiant  SMI  5977  2012-02-16 11:05  fich
```

10.3. Commande chmod: représentation octal:

Syntaxe:

chmod mode fich

- « mode » est un entier écrit sur 3 chiffres de 0 à 7, les chiffres sont écrits en octal.

- Le premier chiffre correspond aux droits d'accès pour le propriétaire.
- Le deuxième chiffre correspond aux droits d'accès pour le groupe.
- Le troisième chiffre correspond aux droits d'accès pour les autres utilisateurs.

- L'écriture binaire (en base 2) de chaque chiffre définit les droit d'accès pour chaque utilisateur.

octal	**binaire**	**droits**
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Exemple1: Ecrire la commande qui exprime pour le propriétaire les droits de lire, d'écrire et d'exécuter, pour le groupe les droits de lire et d'écrire et pour les autres utilisateurs les droits de lire et d'exécuter.

% chmod 765 fich

% ls -l fich

-rwx rw- r-x 1 etudiant SMI 5977 2012-02-16 11:05 fich

En effet:

7 = rwx représente les droits pour le propriétaire.

6 = rw- représente les droits pour le groupe.

5 = r-x représente les droits pour les autres.

Exemple2: Ecrire la commande qui exprime pour le propriétaire le droit de lire, pour le groupe les droits de lire et d'exécuter .

%chmod 450 fich

%ls -l fich

- r-- r-x --- 1 etudiant SMI 5977 2012-02-16 11:05 fich

En effet:

4 = r-- représente les droits pour le propriétaire.

5 = r-x représente les droits pour le groupe.

0 = --- représente les droit pour les autres utilisateurs.

Remarques sur les protections

- Tous les répertoires inclus dans le chemin d'accès d'un fichier doivent être accessibles en exécution pour qu'il puisse être atteint.
- Lien dur: On peut modifier les droits d'accès d'un fichier à partir du fichier lien.
- Lien symblique: On peut modifier les droits d'accès d'un fichier à partir du lien symbolique:

  N.B. Le chmod sur un lien symbolique ne modifie pas les droits du lien mais modifie les droits du fichier pointé.

11. Recherche de fichiers : La commande « find »

Permet de chercher dans un répertoire (repère de base) et dans toute l'arborescence depuis ce répertoire (on explore récursivement le répertoire et ces sous répertoires), les fichiers vérifiant certains critères.

Syntaxe:

find repertoires critères options

- Si le critère n'est pas spécifié, alors la commande « find » affiche récursivement tous les répertoires et fichiers du répertoire précisé dans la commande.
- L'option « -print ». Elle est implicite sur la plupart des systèmes Unix) . Elle permet d'afficher les résultats de la recherche sur la sortie standard (écran).
- L'option **« -name »**: fait une recherche sur le nom de fichier. On peut utiliser les métacaractères.

Exemples

- Recherche de tous les fichiers nommés « test.c » à partir du répertoire « home».

  % find /home -name test.c -print
- Afficher tous les fichiers qui termine avec la chaine « .txt » à partir du répertoire courant, c'est à dire toutes les noms des fichier « *.txt », dans ce cas, il faut mettre le critère entre

guillemets (double quottes) ou entre quottes. entre guillemets

% find . -name "*.h" -print ou % find . -name '*.h' -print

Remarque:

- Si le répertoire est précisé avec un chemin relatif, les résultats sont aussi affichés avec un chemin relatif.
- Si le répertoire est précisé avec un chemin absolu, les résultats sont aussi affichés avec un chemin absolu.

Exemples

- Cas du chemin relatif.

```
% find . -print
.
./test1.c1
./cours
./cours/seance1
./cours/seance2
./test2.c
```

- Cas du chemin absolu

```
% find  /home/etudiant  -print
/home/etudiant
 /home/etudiant /test1.c1
 /home/etudiant /cours
 /home/etudiant /cours/seance1
 /home/etudiant /cours/seance2
 /home/etudiant /test2.c
```

Options « -iname »

L'option « -iname »: fait la recherche suivant le numéro d'inode.

Exemple: Supposons que le fichier « fiche », localisé dans le sous-répertoire « cours », a pour numéro d'inode « 1391 ».

```
% find . -inum 1391 -print
./cours/fiche
%
```

Options« -type »

L'option « -type » permet de faire une recherche par type de fichier. « type » prend les valeurs:

« f »: pour fichier normal

« d »: pour répertoire

« l »: pour lien symbolique

« p »: pour les tubes nommés (les pipes)

Exemple:

```
% find . -name "re*" -type d -print
  ./rep1
  ./rep2
```

Options « -user » et « -group »

Les options « -user » et « -group » permettent de chercher les fichiers et répertoires vérifiant les critères sur le nom du propriétaire et du groupe.

Exemple

- Chercher tous les fichiers, qui ont pour propriétaire «etudiant » et pour groupe « SMI », à partir du répertoire courant .

```
%  find . -type f  -user  etudiant -group SMI -print
./fich1
 ./fich2
./.profile
```

Option « -ls »

L'option « -ls » affiche des informations détaillées sur les fichiers trouvés correspondant au critère au lieu du simple nom de fichier.

Exemple1:

```
% find . -inum 1391 -print
   ./cours/fiche
% find . -inum 1391 -ls
1391 572 -rwxr-xr-x 1 etudiant  SMI 582192  2014-23-10 10:55 ./cours/fiche.
./cours/fiche
```

N.B. 582192 est la taille du ficher « fiche » en octets, alors que 572 est la taille du fichier en kilo octets.

12. La commande « sort »

La commande « sort » permet

- de trier les lignes de caractères saisies sur la ligne de commande.
- de trier les lignes des fichiers passés en argument à la commande « sort ».
- de faire le trie sur un champ particulier.

Par défaut:

- La sortie (le résultat de la commande) est dirigée sur la sortie standard.
- Le trie s'effectue par ordre alphanumérique (selon l'ordre ASCII standard).

12.1. Trie des lignes saisies sur la ligne de commandes

La commande « sort » : lit les lignes saisies sur la ligne de commande, trie ces lignes par ordre alphanumérique, ensuite, affiche sur l'écran les lignes triées.

Pour la saisie des lignes:

- On marque la fin de la ligne par un retour chariot (<CR>) on tape sur la touche « Entree » (ou « return »).
- On marque la fin de la saisie par « ^D » (on tape sur la combinaison des touche « Ctrl » et « d ou D » .

Exemple:

```
% sort
        Mohammed
        Ali
        Fatima
^D
        Ali
        Fatima
        Mohammed
%
```

12.2. Trie des lignes à partir d'un fichier

Syntaxe :

sort [option] fiche

La commande « sort » trie les lignes du fichier « fiche » et affiche le résultat sur la sortie standard.

- L'option « -o » permet de spécifier le fichier dans lequel sera écrit le résultat du tri.

- Par défaut, le trie s'effectue selon l'ordre ASCII standard. Les options suivantes permettent de choisir un autre ordre pour le tri.
  - L'option « - n »: spécifie l'ordre numérique.
  - L'option « -d »: le trie est effectué selon l'ordre défini dans un dictionnaire.
  - L'option « -f »: on ignore majuscules et minuscules.
  - L'option « -r »: renverse l'ordre

Exemple:

- Trier le fichier «/etc/passwd » et afficher les résultats sur la sortie standard.

```
% sort /etc/passwd
```

- Trier le fichier «/etc/passwd » et sauvegarder le résultat (le fichier trié) dans le fichier « test_sort »

```
% sort /etc/passwd  -o  test_trie
```

12.3. Trie sur un champ particulier

La commande « sort » peut trier suivant un champ en utilisant le symbole "+" suivi du numéro du champ.

Syntaxe :

```
sort [option] [-tcar] [-kpos1] [-kpos2] [fichier...]
```

- Par défaut, les champs sont délimités (séparés) par une tabulation ou par espace.
- L'option « -tcar » permet de spécifier un caractère « car » qui désignera le nouveau délimiteur (séparateur) de champs.
- « pos1 » désigne le premier champ et « pos2 » désigne le dernier champ, selon lesquels le trie s'effectuera.
- L'option « -b » ignore les espaces en début de champ

Exemple:

- Effectuer un tri numérique sur le 3ème champ du fichier «/etc/passwd». Le séparateur des champs et le caractère « : ».

```
% sort /etc/passwd -k3 -n -t:
```

- Effectuer un tri, avec ordre inversé, sur le 4ème champ du fichier « /etc/passwd » et écrire le résultat dans le fichier «passwd_tri». Le séparateur des champs et le caractère « : ».

```
% sort -r -k4 -t:  /etc/passwd  -o  passwd_trie
```

Exemple : sort -n -k2 -k2 ou sort -n -k2,2 ou -k2n,2n

- On utilise « 2,2 » ou « 2n,2n » pour indiquer à la commande « sort » de commencer le tri sur la 2ème colonne (ou champs) et de l'arrêter à la 2ème colonne. La lettre « n » indique à « sort » de traiter cette colonne en tant que champ numérique.
- On peut spécifier les séparateurs de champs avec l'option « -t ».

  % sort -t: -k2,2 fiche: trier le fichier « fiche » sur le deuxième champs, où le séparateur de champs est le caractère « : »

  % sort -t"$\t" -k2,2 fiche: trier le fichier « fiche » sur le deuxième champs, où le séparateur de champs est la tabulation.

- Tri sur plusieurs champs : on peut faire un tri multiple en combinant les options. On peut par exemple faire le tri sur la deuxième colonne (champ) ensuite sur la première colonne (champ)
- On peut aussi faire le tri à partir d'un certain caractère d'un champ. Pour cela on spécifie le « champ.pos ». Exemple « sort -k2.3 » commence le tri à partir du troisième caractère du deuxième champ.

13. Expression régulière

Définition: Une expression régulière est une séquence de symboles qui permet d'identifier (décrire) une famille de chaînes de caractères existantes au moyen des métacaractères où un métacaractère est un caractère spécial qui permet de générer un ensemble de possibilités.

Dans une expression régulière, on peut trouver tous les caractères sauf le caractère de saut de ligne (fin de ligne, \n, ASCII(10), line feed).

Métacaractères des Expressions régulières

Attention: il ne faut pas les confondre avec les métacaractères du Shell.

- **« * »:** le caractère pointé par « * », c'est-à-dire le caractère qui précède le symbole « * », peut apparaître 0 ou plusieurs fois.

  Exemple: ab* désigne une chaine qui contient le caractère « a » suivi de 0 (zéro) ou plusieurs caractères « b ».

  Attention: « * » n'a pas la même signification que le métacaractère du shell (génération de non de fichiers).

- **« + »:** le caractère pointé par « + » peut apparaître au moins une fois dan la chaine.

Exemple: ab+ désigne une chaine qui contient le caractère « a » suivi d'au moins une fois le caractère « b ».

- **« ? » :** le caractère pointé par **« ? »** peut apparaître 0 (zéro) ou 1 fois (au plus une fois).

  Exemple:

  ab?c désigne la chaine « abc » ou « ac »

- **« . » :** désigne n'importe quel caractère sauf le caractère line-feed (fin de ligne).

  N.B. le métacaractère « . » est équivalent au métacaractère « ? » du shell.

  Remarque : « * » du shell est équivalent au « .* » des expressions régulières

- **« ^»:** début de ligne.

  Exemple:

  ^abc : signifie que la ligne commence avec chaîne « abc ». La chaine « abc » est en début de la ligne.

- « $ »: fin de ligne.

  Exemple:

  abc$: signifie que la chaine « abc » est en fin de ligne.

- **« \ »:** utilisé pour bloquer (verrouiller) l'interprétation d'un métacaractère. Le métacaractère pointé par « \ » sera considéré comme un caractère et non comme un métacaractère.

  Exemple: « * »: désigne le caractère « * ».

- **[liste]**: désigne un caractère quelconque parmi ceux qui sont décrit dans «liste».

  Exemples:

  [asy] ou [a,s,y] ou ou [a|s|y] : chaîne contenant a ou s ou y

  [a-zA-Z] : chaîne contenant un caractère alphabétique miniscule ou majuscule

  [a-z] [0-9]: chaîne contenant une lettre minuscule suivie d'un chiffre

- **[^liste]:** désigne un caractère quelconque, différent des caractères décrits dans «liste».

  Exemples:

  [^0-9] : désigne une chaîne ne contenant pas de chiffre.

  ^[^0-9] : désigne une chaîne ne contenant pas de chiffre au début de la ligne.

- **[.]** : chaîne contenant le caractère « . » (pas d'interprétation)

14. La commande grep (Global Regular Expression Printer)

La commande « grep » cherche, dans un fichier passé en argument, les lignes qui vérifient le critère de recherche (les lignes qui contiennent l'expression régulière précisée sur la ligne de commande), et affiche les lignes trouvées sur la sortie standard (écran).

Syntaxe :

grep [options] expression-régulière [liste-de-fichiers]

Remarque: Pour éviter toute interprétation du shell des caractères spéciaux, il vaut mieux mettre l'expression régulière entre simple quottes « ‘’ » ou entre doubles quottes «"" ».

Exemple : Recherche la chaine « etudiant » dans le fichier « /etc/passwd ».

```
% grep "etudiant" /etc/passwd
```

Les principales options de grep:

- L'option « - c »: affiche le nombre de lignes trouvées,
- L'option « -i » : ignore les majuscules/minuscules,
- L'option « -l »: liste les fichiers, donnés en argument (en entrée), dans lesquels au moins une ligne a été trouvée,
- L'option « -n »: affiche les lignes trouvées ainsi que leurs numéros.
- L'option "-v »: affiche les lignes qui ne contiennent pas l'expression de recherche.

15. La commande wc

Syntaxe :

wc [option] fiche

La commande "wc" compte et affiche sur la sortie standard:

- Le nombre de lignes du fichier « fiche ». Les lignes sont séparées par un retour chariot (<CR>), lorsqu'on tape sur la touche « Entrée » ou « Return ».
- Le nombre de mots. Les mots sont séparés par des espaces (la barre space) ou par des tabulations (touche tab).
- Le nombre de caractères.

Remarque: Si plusieurs fichiers sont passés sur la ligne de commande alors le comptage se fait par fichier.

Les options de la commande « wc » sont :

- Option « -l » : compte seulement le nombre de lignes,
- Option « -w » compte seulement le nombre de mots,
- Option « -c » compte seulement le nombre de caractères.

Exemples:

- Afficher le nombre de lignes, de mots et de caractères du fichier /etc/passwd

  % wc -l /etc/passwd

- - Afficher le nombre de lignes du fichier /etc/passwd

  % wc -l /etc/passwd

16. La commande cut

La commande « cut » permet de sélectionner (d'extraire) des caractères (colonnes) ou des champs de chaque ligne et les afficher à la sortie standard (écran).

Syntaxe :

- Pour extraire des caractères :

  cut -cliste [fichier]

- Pour extraire des champs, par défaut les champs sont séparés par des tabulations ou des espaces :

  cut -fliste [-d delim] [-s] [fichier]

« liste » spécifie les numéros des caractères ou des champs à sélectionner. La numérotation des caractères ou des champs commence à 1. Les formats de liste:

- « n1,n2,n3 » sélectionne les caractères (ou les champs) n1, n2 et n3.
- « n1-n2 » sélectionne des caractères ou des champs de n1 à n2.
- « n1- » sélectionne tous les caractères (ou tous les champs) à partir du caractère (champs) numéro « n1 » jusqu'à la fin de la ligne.
- « -n1 » sélectionne tous les caractères (ou tous les champs) à partir du début de la ligne jusqu'au caractère (champs numéro n1).

Exemples:

- Extraire de chaque ligne du fichier «fiche», les caractères 1, 3, 5 ainsi que les caractères de 7 à 9.

  % cut -c1,3,5,7-9 fiche

- Extraire de chaque ligne du fichier «facture», les 2 premiers champs ainsi que les champs de 6 à 8.

  %cut -f-2,6-8 fich

Les options de la commande « cut »

- **L'option « -d »:** elle est utilisée pour spécifier le caractère de séparation de champs (délimiteur). Dans ce cas le séparateur par défaut est ignoré..

Exemple:

- Extraire les champs 1à 4 de chaque ligne du fichier « /etc/passwd », où le caractère de séparation des champs (délimiteur) est « : »

  % cut -f-4 -d: /etc/passwd

- Extraire les champs 4 et 6 du fichier « fiche » où « , » est le caractère de séparation des champs.

  % cut -f4,6 -d, fiche

- **L'option "-s":** elle supprime les lignes sans caractère délimiteur. Elle permet d'écarter toutes lignes qui ne contiennent pas le séparateur.

Exemple

% cut -f3,7 -d: -s fiche

V. Redirection des entrées-sorties standards

Lors de l'exécution d'une commande, un processus est créé. Par défaut le processus: lit ses données d'entrée à partir de l'entrée standard (par défaut le clavier) et écrit les résultats et les erreurs dans la sortie standard (par défaut l'écran). Ces canaux de communications (canneaux d'entrée :sortie) sont désignés par les fichiers :

- « stdin » : pour l'entrée standard, identifiée par l'entier 0.
- « stdout »: pour la sortie standard, identifiée par l'entier 1.
- « stderr »: pour la sortie d'erreur standard, identifiée par l'entier 2.

Il est possible de rediriger les entrées/soties standards des commandes afin que la lecture se fasse à partir d'un fichier et que l'écriture se fasse dans fichier.

1. Redirection de l'entrée standard

On peut rediriger l'entrée standard d'une commande afin que la lecture se fasse à partir d'un fichier grâce au symbole« < ».

Syntaxe :

commande < fichier_entree

La commande effectue ses entrées à partir du fichier «fichier_entree »

Exemple

% cat < donnees_entree

La commande lit le fichier «donnees_entree » et l'affiche sur la sortie standard, au lieu de lire les données au clavier.

2. Redirection de la sortie standard

On peut rediriger la sortie standard d'une commande afin que l'écriture se fasse dans un fichier grâce au symbole « > » ou « >> ».

Syntaxe :

commande > fichier_sortie

commande >> fichier_sortie

- Si le fichier « fichier_sortie » n'existe pas, alors il sera créé.
- Si le fichier « fichier_sortie » existe, alors:
  - Si le symbole « > » est utilisé alors le fichier « fichier_sortie » sera écrasé et remplacé par la nouvelle sortie standard de la commande.
  - Si le symbole « >> » est utilisé, alors la nouvelle sortie standard sera rajoutée à la fin du fichier « fichier_sortie ».

Exemple:

- On trie le fichier « fiche1 » et on écrit le résultat dans le fichier « fiche_trie » au lieu d'afficher le résultat sur l'écran.

  % sort fiche1 > fiche_trie

- On trie le fichier « fiche2 » et on écrit le résultat à la fin du fichier « fiche_trie ».

  % sort fiche2 >> fiche_trie

- Dans l'exemple suivant on montre qu'on peut rediriger l'entrée et la sortie en même temps.

  % sort < fiche > fichier_trie

3. Redirection de la sortie d'erreurs standard (stderr)

Lors de l'exécution d'une commande, des messages d'erreurs peuvent être affichés à l'écran si un problème survient en cours d'exécution. On peut rediriger la sortie des messages d'erreurs dans un fichier. Ceci permet d'éviter d'avoir un mélange entre les résultats normaux et les messages d'erreurs dans une même sortie (standard ou fichier). Ceci peut se faire grâce aux chaines « **2>** » ou « **2>>** ».

Syntaxe :

commande 2> fichier_sortie

commande 2>> fichier_sortie

Comme pour la redirection de la sortie standard,

- si le fichier sorti n'existe pas alors il sera créé.
- si le fichier sorti existe alors:
  - si on utilise la chaine **« 2> »** alors le fichier sera écrasé et remplacé par la nouvelle sortie standard.
  - si on utilise la chaine **« 2>> »** , alors la sortie standard se rajoute à la fin du fichier sortie.

Exemple:

On exécute la commande « find » sans arguments. Les résultats de la recherche sont redirigés ver les le fichier «res_find» alors que les messages d'erreurs (par exemple on ne peut pas explorer un répertoire protégé en lecture) seront redirigés vers le fichier « erreur_find ».

```
% find . > res_find  2>erreur_find
% find . > >res_find  2>>erreur_find
```

Remarque: Pas d'espace entre le "2" et le ">"

VI. Les tubes (Les pipes)

Les pipes (tubes) permettent de rediriger la sortie d'un processus vers l'entrée d'un autre. Les processus s'exécutent en parallèles.

Syntaxe :

cmdA | cmdB

- Le symbole « | » appelé « pipe ». IL permet de relier deux commandes entre elles.
- La sortie standard de la commande « cmdA » est utilisée comme entrée standard de la commande « cmdB ».

Exemple

- Exécuter la commande « ls -l » et rediriger la sortie de la commande vers le fichiers « resultat.txt »

```
%  ls -l > resultat.txt
```

- Compter le nombre de lignes, le nombre de mots et le nombres de caractères du fichier « resultat.tx ». Et rediriger le résultat vers le fichier « cp.txt »

  % wc < resultat.txt > cp.txt ou % wc resultat.txt > cp.txt

- Les deux commandes peuvent être combinées pour élaborer une seule:

  % ls -l | wc > cp.txt

Exemple

```
% ls -l | wc
    16    124    921
% ls -l | wc | wc
    1     3      24
```

1 désigne le nombre de lignes,

3 désigne le nombre de mots,

24 désigne le nombre de caractères.

VII. Contrôle des processus

Le shell permet de contrôler les processus qui s'exécutent. Un processus s'exécute soit en foreground (avant plan), soit en background (arrière plan ou en tâche de fond).

1. Exécution en foreground

Par défaut, quand on exécute une commande, elle s'exécute en avant plan. Tant que l'exécution de la commande n'est pas terminée, on ne peut pas lancer une nouvelle commande, avec le même processus shell. Pour lancer une nouvelle commande, on doit ouvrir un autre terminal et par conséquent lancé un nouveau processus shell, dans ce cas la nouvelle commande peut être exécutée avec le nouveau processus shell.

Remarques:

- Un processus lancé en avant plan peut être suspendu (bloqué) en tapant la combinaison des touches « Ctrl-Z ».

  Attention: L'exécution du processus est suspendue mais le processus est toujours actif.
- Un processus bloqué, peut être relancé en avant plan avec la commande « fg ».
- Un processus bloqué, peut être relancé en arrière plan avec la commande « bg ».
- Un processus lancé en avant plan, peut être interrompu en tapant la combinaison des touche « Ctrl-C ».

2. Exécution en background

Pour que le shell puisse lancer deux commandes en parallèle, c'est à dire lancer une nouvelle commande sans attendre la fin d'exécution de la commande précédente, on doit mettre le symbole « & » (le « et » commercial) après la première commande.

⇒ Dans ce cas, le processus shell crée un processus fils qui exécute la première commande en arrière plan (background) alors que le processus shell (le processus père) s'occupe de l'exécution de la nouvelle commande.

Syntaxe :

nom_cmd &

- « nom_cmd » désigne le nom de la commande à lancer en arrière plan.
- Le symbole « & » permet de lancer la commande « nom_cmd » en arrière plan (background).
- Lorsque la commande est lancée en arrière plan, le shell:
  - Affiche entre crochets, « [num_processus] », le numéro du processus lancé en arrière plan, par le même processus shell.
  - Affiche Le PID (Process IDentifier), le numéro identifiant le processus lancé en arrière plan.
  - Ensuite, affiche le prompt (l'invite de commande) et attend une nouvelle commande.

Exemple : Faire une recherche en arrière plan de tous les fichiers nommés «*.c», à partir du répertoire racine « /», et écrire le résultat de la recherche dans le fichier « resultat ».

```
% find / -name  "*.c"  -type f  -print > resultat &
  [3] 2812
%
```

- Le processus « find » s'exécute en arrière plan.
- Après le lancement de la commande « find » en arrière plan, les informations suivantes sont affichées:
- [3] désigne le numéro du processus s'exécutant en arrière plan, dans ce cas c'est le troisième lancé par le même processus shell.
- L'entier 2812 représente le PID du processus qui s'exécute en arrière plan.
- Ensuite, le shell se met en attente d'une nouvelle commande. La nouvelle commande s'exécute en parallèle avec la première commande.

Question: Que se passe t-il si on écrit

```
% find / -name  "*.c"  -type f  -print  &  > resultat
```

Remarques

- La commande « fg » permet de relancer le processus en avant plan.
- On ne peut pas arrêter (interrompre) un processus en arrière plan avec « CTRL-C ».
- La commande « jobs » permet de lister les processus lancés en arrière plan, par le même processus shell.

  Exemple:

```
% jobs
[1]+  Running                 gedit &
%
```

- Si l'exécution en arrière plan, s'est bien passée, une ligne « Done » apparait indiquant que le processus est terminé.

  Exemple:

```
%
[1] + Done      find / -name  "*.c"  -type f  -print > resultat
```

- Si l'exécution en arrière plan, ne s'est pas bien passée, une ligne « Exit » apparait indiquant que le processus est terminé.

  Exemple:

```
%
[1] + Exit   find / -name  "*.c"  -type f  -print > resultat
```

Précaution:

- Si le processus lancé en arrière plan fait des saisies interactives alors le shell peut interpréter ces saisis comme de nouvelles commandes.

  ⇒ éviter les saisies interactives dans les processus lancés en arrière plan.
- Si le processus lancé en arrière plan fait des affichages sur écran alors ces affichages peuvent se chevaucher avec les sorties standard des nouvelles commandes lancées avec le shell.

  ⇒ éviter les sorties standards dans les processus lancés en arrière plan.

Attention: Un processus ne peut exister que si son père existe. Quand on quitte le processus père alors on quitte automatiquement tous ses processus fils.

⇒ Ne pas quitter le shell alors que ses processus fils s'exécutent en arrière plan.

3. La commande « bg » (Back Ground: arrière plan):

La commande « bg » permet de réactiver un processus suspendu par « CTRL-Z ». L'exécution du processus réactivé s'effectue en arrière plan.

Syntaxe :

bg %num_process

- « num_process » désigne le numéro du processus lancé en arrière plan.
- Pas d'espace entre % et « num_process »

Exemple :

- On lance en avant plan la commande « find ». Pendant l'exécution, on tape la commande «Ctrl-Z» pour suspendre l'exécution du processus.

```
% find / -print > resultat
```

- Lorsqu'on tape « Ctrl-Z », la commande « find » est suspendue, le message suivant est affiché.

```
[1] + Stopped find / -print > resultat
```

- On peut réactiver le processus et relancer, en arrière plan, la suite de la commande « find » en utilisant la commande «bg».

```
% bg %1
[1] 1234 find / -print > resultat &
%
```

4. La commande « fg » (foreground : avant plan)

La commande « fg » permet de :

- de réactiver un processus suspendu par « CTRL-Z ». L'exécution du processus réactiver s'effectue en avant plan.
- de refaire passer en avant plan une commande qui s'exécute en arrière plan (commande lancée avec le symbole « & »).

Syntaxe :

fg %num_process

- « num_process » désigne le numéro du processus lancé en arrière plan ou suspendu par « Ctrl-Z ».
- Pas d'espace entre % et « num_process »

Exemple :

- On lance en avant plan la commande « find ». Pendant l'exécution, on tape la commande «Ctrl-Z» pour suspendre l'exécution du processus.

```
%  find / -print > resultat
```

- Lorsqu'on tape « Ctrl-Z », la commande « find » est suspendu , le message suivant est affiché.

```
[1]  +  Stopped     find / -print > resultat
```

- On peut réactiver le processus et relancer, en avant plan, la suite de la commande « find » en utilisant la commande «fg».

```
% fg  %1
find / -print > resultat
```

6. Commande "wait" sans arguments

La commande « wait » permet faire attendre que tous les processus lancés en arrière plan soient terminés.

Syntaxe :

wait

Exemple:

```
% cat ./SMIS3/*  >  res1.txt    2>erreur1.txt &
% find /  >res2.txt   2>erreur2.txt  &
% wait
% echo " les deux commandes sont terminées «
```

La commande « wait » est lancée en avant plan. On ne peut plus lancer de nouvelles commandes, tant que les commandes lancées en arrière plan n'ont pas terminé.

7. Commande « ps »

La commande ps (Process Status) permet de lister les processus qui sont en cours même sur la machine.

Syntaxe:

ps [options]

La commande « ps » sans options, n'affiche que les processu en cours qui sont lancés par l'utilisateur sur le terminal actuel (par le processus shell).

Exemple:

%ps

PID	TTY	TIME	CMD
2254	pts/1	00:00:00	bash
2705	pts/1	00:00:01	gedit
2710	pts/1	00:00:00	ps

Quelques options:

Pour plus d'option utiliser la commande « man » ou exécuter la commande « ps --help ».

- L'option « -f »

% ps -f

UID	PID	PPID	C	STIME	TTY	TIME	CMD
Etudiant	1937	1935	0	06:43	pts/0	00:00:00	bash
etudiant	2061	1937	0	06:53	pts/0	00:00:00	ps -f

- L'option « u »

% ps u

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	CMD
edtudiant	2214	0.0	0.1	6464	3752	pts/0	Ss+	07:09	0:00	bash
smi	2238	0.0	0.1	6464	3748	pts/2	Ss+	07:10	0:00	bash
etudiant	2263	0.0	0.1	6464	3752	pts/3	Ss	07:16	0:00	bash
etudiant	2351	0.0	0.0	2640	1016	pts/3	R+	07:35	0:00	ps u

Signification des différentes colonnes

- UID (User ID): identifiant de l'utilisateur qui a lancé le processus.
- PID (Process ID): correspond au numéro du processus
- PPID (Parent Process ID): correspond au PID du processus parent.
- STIME (Start TIME): correspond à l'heure du lancement du processus.
- START: idem STIME.
- TTY : correspond au nom du terminal depuis lequel le processus a été lancé.
- TIME : correspond à la durée de traitement du processus.
- CMD : correspond au nom du processus (la Commande exécutée).
- %CPU: facteur d'utilisation du cpu exprimé en %.
- C: idem %CPU
- %MEM : rapport d'utilisation de la mémoire

- RSS: mémoire physqiue (non virtuelle) utilisée par la processus, exprimée en kiloBoctets .
- VSZ : mémoire virtuelle.
- STAT: état du processus

Quelques différents états:

- R (Running): prêt à être exécuté (il sur la file d'attente des processus).
- S (Sleeping): bloqué en attente d'une action externe (par exemple le processus attent une lecture sur la clavier).
- T : processus arrêté, par exemple par Ctrl-Z.

8. Arrêt d'un processus / signaux

Pour arrêter un processus qui s'exécute en arrière plan, on utilise la commande « kill »qui consiste à envoyer des signaux au processus. Le signal est un moyen de communication entre les processus.

Syntaxe :

kill [-num_signal] PID [PID ...]

Le principe de la commande consiste à envoyer un signal (une requête) au processus spécifié dans la commande par son PID. Ce dernier intercepte le message et réagit en fonction du signal envoyé.

Remarque:

- Seulement le propriétaire du processus ou le super-user (utilisateur "root").
- Chaque signal a un nom et un numéro qui dépendent du système utilisé. La commande « kill -l » permet d'obtenir la liste des signaux.

Par défaut, la commande « kill » envoie le signal demandant au processus spécifié de se terminer normalement.

Exemple.

```
% gedit test.c &
  [1] 1234
% kill 1234
  [1] + Terminated  gedit test.c
```

Quelques signaux

- Le signale numéro 1 (SIGHUP): Ce signal est envoyé par le père à tous ses enfants lorsqu'il se termine.
- Le signal numéro 2 (SIGINT): ce signal permet l'interruption du processus demandé (Ctrl+C).
- Le signal 9 (SIGKILL): ce signal permet de forcer l'arrêt du processus.

Exemple:

```
% gedit test.c &
  [1] 1234
% kill  -9  1234
  [1] + Processus arrêté  gedit test.c
```

VIII. Le code retour d'une commande

Le code retour (le statut) d'une commande est un entier positif ou nul, toujours compris entre 0 et 255, retourné par la commande au shell qui permet de signaler à l'utilisateur l'état de l'exécution de cette commande , c'est-à-dire si l'exécution de cette commande s'est bien passé ou non. Par convention:

- un code retour égal à 0 signifie que la commande s'est exécutée correctement.
- Un code différent de 0 signifie soit une erreur d'exécution, soit une erreur syntaxique.

Le caractère spécial « ? » du shell (à ne pas confondre avec le caractère générique « ? » du shell) contient le code retour de la dernière commande exécutée. Pour accéder au contenue de la variable « ? » on utilise le symbole « $ ».

Exemple: Code retour de la commande « grep »

- Si le code est égal à 0: au moins une ligne a été trouvé
- Si le code est égal à 1: aucune ligne n'a été trouvée.
- Si le code est égal à 2: problème d'exécution (mauvaise syntaxe, fichier non accessible, ….)

```
%grep "while"  test.c
%echo $?
%
```

% echo $?

affiche le code de retour de la dernière commande effectuée.

IX. Composition des commandes

1. Le symbole « ; »

Le symbole « ; » permet d'écrire une séquence de commande.

⇒ On peut lancer plusieurs commandes sur la même ligne de commande, dans ce cas les commandes doivent être séparées par le symbole « **;** »

Syntaxe:

cmd1 ; cmd2 ; …

Le shell attend que la commande «cmd1» soit terminée avant de passer à la commande «cmd2». De manière générale, le shell attend que la commande qui précède le symbole « ; » soit terminée avant de passer à la commande suivante.

⇒ L'exécution des commandes s'effectue de gauche vers la droite.

Exemple: Lister le fichier « test.c » s'il existe ensuite compter le nombre de ses lignes.

```
% ls -f test.c; wc -l test.c
```

2. Le symbole « & » : (voir avant)

Syntaxe :

nom_cmd &

Le shell exécute la commande qui précède le symbole « & » en arrière plan et passe à la commande suivante.

3. Les séquences ( ) et { }

Ils servent regrouper une ou plusieurs commandes. Cet ensemble des commande est appelé une macro-commande.

(suite_cmds) : Pour les commandes entre parenthèses, le processus shell crée un processus fils pour exécuter ces commandes.

⇒ Ces commandes sont exécutées dans un sous-shell

{ suite_cmds **;** } : Pour les commandes qui sont entre accolades, c'est le processus shell lui-même qui les exécute.

NB. Laisser un espace après "{".

Exemple1: Considérons la macro-commande (cd cours; pwd):

```
% pwd
  /home/etudiant
% (cd cours; pwd)
  /home/etudiant/cours
% pwd
  /home/etudiant
%
```

En effet, toutes les commandes ne sont pas exécutées par le même shell:

- les commandes « cd cours » et « pwd », qui sont entre accolades, sont exécutées par un processus fils du shell courant. Après exécution, on retourne vers le shell courant.
- la dernière commande « pwd » est exécutée dans le shell courant et non par le processus shell fils.

Exemple2: Considérons la macro-commande {cd cours; pwd}:

```
% pwd
  /home/etudiant
% { cd cours; pwd; }
  /home/etudiant/cours
% pwd
   /home/etudiant/cours
```

Toutes les commandes sont exécutées par le même shell.

4. Exécution conditionnelle

Opérateurs « && »

Syntaxe:

```
cmd1 && cmd2
```

Exécute la commande «cmd1». Puis, si l'exécution de la commande « cmd1 » s'est bien passée , c'est-à-dire la commande «cmd1» a un code retour (un statut de sortie) égal à 0 alors on exécute la commande «cmd2».

Exemple:

```
% ls -l > fiche  && wc test.c
    11   17   118  test.c
% ls –j > fiche  && wc test.c
    ls: option invalide –'j'
```

Dans ce cas la commande « wc » n'a pas été exécutée.

Opérateurs « || »

Syntaxe:

cmd1 || cmd2

Exécute la commande «cmd1» . Si la commande «cmd1» a un code retour différent de 0 alors on exécute la commande «cmd2».

Exemple:

```
% ls -l > fiche  || wc test.c
% ls –j > fiche  ||  wc test.c
ls: option invalide –'j'
11   17    118   test.c
```

Dans ce cas la commande « wc » a été exécutée.

Chapitre 3

La programmation Shell

I. Variables du shell

1. Notion de variable

Une variable est une donnée identifiée par un nom. Le nom d'une variable est une suite de caractères alphanumérique.

Attention: Pour les noms de variables, le shell fait la différence entre les MAJUSCULES et les minuscules.

Remarque: Il n'y a pas de notion de typage en shell, toutes les valeurs affectées aux variables sont des chaînes de caractères, et par conséquent:

- il n'y a pas de déclaration de variables.
- il n'y a pas de valeur numérique.

Sous UNIX, on distingue deux types de variables, les variables locales (les variables définie par l'utilisateur), et les variables d'environnement (variables globales ou exportées).

- Une variable locale est spécifique au processus shell en cours (le shell dans lequel la variable est définie), seul ce processus pourra l'exploiter et elle ne sera pas disponible pour tous les processus fils créés.
- Les variables d'environnements sont transmises aux sous-shells c'est-à-dire aux processus lancés par le shell (les processus fils du shell).

2. Affectation des valeurs aux variables

Pour affecter une valeur (une chaine de caractères) à une variable on utilise le symbole « = »

Syntaxe:

var=val

Affecte la valeur « val » à la variable « var »

Attention:

- Il n'y a pas d'espaces avant et après le symbole «=».
- Si « val » contient des espaces ou des caractères spéciaux il faut la mettre en double quotes (voir plus loin).

Exemple:

```
% mois=janvier
%
```

Affecte la chaine "janvier" à la variable « mois »

3. Utilisation des variables

Pour accéder à la valeur (au contenu) d'une variable, on la précède par le symbole « $ ». Le symbole « $ » sert à substituer (remplacer) la variable par sa valeur.

Syntaxe:

```
$nom_variable
```

Attention: Il n'y a pas d'espaces entre le symbole « $ » et la variable.

Exemple:

```
% echo $mois
  janvier
%
```

La commande « echo » permet d'afficher ses arguments sur la sortie standard (écran).

Il est parfois nécessaire de mettre la variable entre les accolades:

```
${nom_variable}
```

Exemple:

```
%vol=volume
%echo  $vol60

%
```

N'affiche rien, en effet, pour le shell, « vol60 » est une variable non initialisée (ne contient aucune valeur). Si on veut afficher «volume60», on doit rajouter les accolades.

```
% echo ${vol}60
  volume60
%
```

4. Destruction d'une variable

La commande « unset » permet de détruire (supprimer) une variable (la variable ne sera plus accessible)

Syntaxe:

unset nom_variable

Exemple:

```
% unset mois
% echo $mois

%
```

N'affiche rien

5. Commande « readonly » : protection d'une variable

Pour protéger la modification du contenu d'une variable on utilise la commande « readonly ».

Syntaxe:

readonly nom_variable

Exemple:

```
% readonly mois
% mois=Mars
  bash: mois : variable en lecture seule
%
```

Remarque: Une variable protégée en écriture (une variable en lecture seule) même vide, on ne peut plus ni la rendre modifiable ni la détruire sauf si on quitte le shell.

6. Variables alias

- La commande « alias »

Un alias est une variable qui sert à redéfinir le nom d'une commande ou à lui donner un nouveau nom. Ceci permet de donner des noms simples (abrégés) à des commandes.

Syntaxe:

alias nom=commande

Attention: Il n'y a pas d'espace avant et après « = »

Exemple:

- Donner un nouveau nom à la commande « ls -l » en lui donnant un nom simplifié (nom abrégé)

```
% alias ll="ls -l"
% ll
  -rwxr--r-- 1 etudiant etudiant 66 2014-11-23 17:29 Progdebut
```

- Redéfinir le nom de la commande « ls» afin qu'elle désigne la commande «ls -a».

```
% alias ls="ls –a"
% ls
. .. ProdDebut
```

La commande « alias » sans arguments, affiche la liste des alias définis dans le même shell par l'utilisateur.

Exemple:

Afficher la liste des alias

```
% alias
  ll="ls -l"
  ls="ls –a"
%
```

- La commande « unalias »

Pour rendre une variable alias inaccessible (détruire la variable alias), on utilise la commande « unalias ».

Exemple:

- Détruire la variable alias « ll »

```
% unalias ll
%ll
  ll : commande not found
```

7. Exportation de variables : commande « export »

Les variables définies par l'utilisateur, sont des variables locales. Elles ne sont connues que par les processus shell qui les a crées. Elles ne sont pas transmises aux processus fils.

Remarque: Pour les tests, on utilise la commande « bash » qui permet de créer un sous-shell (processus fils du shell). Pour quitter le sous-shell, on exécute la commande « exit ».

Exemple 1: On teste de la commande « bash »

% ps -f

UID	PID	PPID	C	STIME	TTY	TIME	CMD
etudiant	2128	1956	0	16:08	pts/1	00:00:00	bash
etudiant	2550	2128	0	16:32	pts/0	00:00:00	ps –f

%bash

% ps –f

UID	PID	PPID	C	STIME	TTY	TIME	CMD
etudiant	2128	1956	0	16:08	pts/1	00:00:00	bash
etudiant	2600	2128	0	16:35	pts/1	00:00:00	bash
etudiant	2620	2600	0	16:36	pts/1	00:00:00	ps –f

%exit

Exemple 2: variable locale

- Considérons la série de commandes suivantes:

```
% a=35; echo $a
  35
%bash
%echo $a

%
```

N'affiche rien

En effet :

- Les commandes « a=35 » et « echo $a » sont exécutées dans le shell courant.
- La commande « bash » permet de créer un sous-shell.

- La dernière commande « echo $a » est exécutée dans le sous-shell (le processus fils du shell créé par la commande « bash »).
  - puisque la variable « a », définie dans le shell père, n'est pas transmise au sous-shell («a» est une variable locale).
  - et puisque la variable « a » n'est pas définie dans le sous-shell, alors la dernière commande n'affiche rien.

Pour que la variable locale soit globale entre le processus père et ses processus fils, il faut l'exporter vers l'environnement, par conséquent la variable locale devienne une variable d'environnement (variable globale). Pour exporter une variable on utilise la commande « export ».

Syntaxe:

export nom_variable

export nom_variable =valeur

Permet de déplacer la variable « nom_variable », définie dans la zone des variables locales du processus shell, vers la zone d'environnement (la zone des variables globales).

Exemple 3:

```
% a=35
% export a
%bash
%echo $a
  35
%
```

En effet

- Les commandes « a=10 » et « export a » sont exécutées dans le shell courant.
- La commande « bash » permet de créer un sous-shell.
- La dernière commande « echo $a » est exécutée dans le sous-shell. Puisque la variable « a » est exportée, donc elle est accessible par le sous shell.

Attention: Exportation du père vers les fils et non des fils vers le père.

Exemple 4:

```
%(B=20; export B)
%echo $B
%
```

En effet:

- Les commandes (B=20; export B) sont exécutées dans un sous-shell, par conséquent, la commande « export B » est exécutée dans le sous-shell.
- La commande « echo $B » est exécutée dans le shell courant (le processus père).

8. Les variables d'environnements usuelles.

L'environnement du shell est initialisé par le fichier « /etc/profile », seul l'administrateur système qui a le droit de modifier ce fichier. A chaque lancement d'un processus shell, des variables prédéfinies sont initialisés. Ils servent à configurer l'utilisation du Shell et les outils Unix: ce sont les variables d'environnement usuelles (prédéfinies).

Quelques variables d'environnement usuelles

HOME : référence le répertoire du login.
PATH: référence le chemin de recherche pour l'exécution des commandes
CDPATH: référence le chemin de recherche pour la commande cd
PS1: référence l'invite de commande (le prompt).
LANG : référence la Langue utilisée pour les messages.
SHELL : référence Le shell de l'utilisateur
HISTSIZE : Nombre de commandes à mémoriser dans l'historique.
USER : référence Le login utilisateur.
PWD : référence Le répertoire courant.
SECONDS: le nombre de secondes écoulés depuis le lancement du script.
RANDOM: génère un nombre aléatoire entre 0 et 32767 à chaque appel.

La commande « env » permet d'afficher les variables d'environnement (y compris celles définies par l'utilisateur).

```
% env
        USER=mdaoudi
        HOME=/users/mdaoudi
        PWD=/home/mdaoudi
        PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin/:/usr/bin:/sbin:/bin:  /usr/games
        SHELL=/bin/bash
        LANG=fr_FR.UTF-8
        …
%
```

9. Modification des variables d'environnement usuelles:

Comme exemple, on travaillera avec la variable d'environnement « PATH ». Cette variable référence la liste des répertoires que le shell explore pour chercher la commande à exécuter. Par exemple: si « PATH » contient les chemins:

/bin/:/usr/bin:/usr/local/bin:

alors pour exécuter la commande « cat », le shell va la chercher dans ces répertoires:

/bin/cat, /usr/bin/cat, /usr/local/bin/cat:

Exemple : Soit l'exécutable « test» d'un programme écrit en langage « C ». Supposons qu'il se trouve dans le répertoire courant. Pour exécuter ce programme , on tape la commande

```
%./test
```

Si le « PATH » contient le répertoire courant (le répertoire "."), alors pour exécuter une commande, le shell cherche aussi dans le répertoire courant. Dans ce cas, pour exécuter le peogramme « test », il suffit de taper:

```
%test
```

Au lieu de

```
%./test
```

On peut modifier la valeur de la variable « PATH », par exemple pour lui rajouter « . », de la manière suivante

```
%PATH=$PATH:.
```

Si le shell en cours (le shell courant) lance un sous-shell, alors le sous-shell héritera des variables d'environnement. Dans ce cas la modification portée sur « PATH » sera pris en compte par les sous-shell, mais elle ne sera pas prise en compte par un autre sou-shell (par exemple dans un autre terminal).

Pour modifier une variable d'environnement de façon permanente (prise en compte par tous les processus shell lancés), on peut ajouter une ligne dans le fichier d'initialisation du shell (~/.bashrc pour Bash)

Dans le fichier « ~/.bashrc »

```
#~/.bashrc : excutable bay bash(1) for non-login shells
….
# on rajoute la ligne suivante
PATH=$PATH:.
```

Pour mettre à jour l'environnement (c'est-à-dire la modification soit pris en compte) en tape

```
% . ./bashrc
```

II. Les étapes de l'interprétation d'une ligne de commandes

1. Introduction

Le shell lit, dans l'ordre suivant, la ligne de commande avant d'exécuter la commande.

- substitution de variables : le shell remplace les variables par leurs valeurs
- substitution de commandes : le shell remplace une variable par son contenu qui est le résultat d'une commande.
- interprétation des pipes et des redirections.
- interprétation des caractères spéciaux pour la génération des noms des fichiers.
- exécution de la commande

Les arguments d'une commande peuvent contenir des caractères spéciaux qui seront interprétés par le shell avant d'être interprété par la commande. Dans ce cas on doit indiquer au shell d'interpréter ou non ces caractères spéciaux.

2. Rappel des caractères spéciaux du shell déjà vus:

- Le caractère « $ » : sert pour accéder au contenu d'une variable (sert à substituer la variable par sa valeur).
- Les caractères « ^ », « ? », « [ ] » et « * » : servent à la génération des noms de fichiers par le shell.
- Les caractères « < », « > », « >>», « << » et « 2> »: servent à la redirection des entrées/sorties.
- Le caractère « | »: ce caractère est utilisé dans les pipes.
- Les caractères « & », « ; », « && » et « || » : servent pour la composition de commandes.
- Le caractère back-slash « \ »: C'est le caractère d'échappement (verrouillage des caractères). Il permet d'annuler l'interprétation, par le shell, du caractère qui vient juste après. Le caractère spécial pointé par « \ » sera considéré comme un caractère et non comme un caractère spécial.

Exemple:

```
%a=20
% echo  $a
  20
% echo \$a
  $a
```

« \$ » annule l'interprétation du caractère spécial « $ » qui consiste à accéder au contenu de la variable « a ». Donc la command renvoie la chaine "$a".

3. Les caractères simples quotes « ' »:

Ils sont utilisés pour annuler l'interprétation de tous les caractères spéciaux entre des simples quotes à l'exception de la simple quote.

Exemple 1 :

```
%a=20
% echo ' a=  $a  >  res '
  a=  $a  >  res
 %
```

En effet:

- Le caractère spécial « $ » à l'intérieur des simples quotes a été ignoré (pas de substitution).
- Le résultat n'a pas été redirigé vers « res », car le caractère spécial « > » a été ignoré.

Exemple 2 :

```
%a=20
% echo ' a=  '$a  >  res ''
% cat res
  a=  20
```

En effet:

Les interprétations des caractères spéciaux « $ » et « > » n'ont pas été ignorés, car ils sont mis entre simples quotes à l'intérieurs des simples quotes.

4. Le caractère double quotes « " » :

Ils sont utilisés pour annuler l'interprétation de tous les caractères spéciaux entre des doubles quotes à l'exception des caractères spéciaux**:** « \ », « " », « $ » et « ` » (back quote)

Exemple 1:

```
%a=20
% echo " a=  $a  >  res "
   a=  20  >  res
  %
```

En effet:

Le caractère « $ » a été interprété par contre le caractère spécial « > » a été ignoré.

Exemple 2:

```
%a=20
% echo " a=  $a  "> res ""
% cat res
  a=  20
```

En effet:

L'interprétation du caractère spécial « > » n'a pas été ignoré, car il est mis entre double quotes à l'intérieurs des doubles quotes.

5. La substitution de commandes

La substitution de commandes consiste à remplacer le nom d'une commande par son résultat. Il en existe deux formes : **$(commande)** ou **`commande`**

- La forme avec les caractères back-quotes

Les caractères back-quotes « ``» (les accents graves, s'insèrent avec Alt Gr + 7 sur un clavier AZERTY français).

- Ils permettent de substituer le résultat d'une commande écrite entre des back-quotes, dans une chaine de caractères.

Exemple1:

```
% echo la date est  `date`
La date est  mardi 25 novembre 2014, 11:59:10
```

- Ils permettent d'assigner la sortie d'une commande à une variable ou d'utiliser la sortie d'une commande comme un argument d'une autre commande.

Exemple 2:

```
% x=`ls`
% echo $x
  ProgDebut   res
% rep=`pwd`
% echo $rep
  /home/etudiant/cours
%cd /
%cd  $rep ; pwd
  /home/etudiant/cours
```

- La forme $(commande)

Elle permet de substituer le résultat d'une commande dans une chaine de caractères.

Exemple3:

```
% echo la date est  $(date)
la date est  mardi 25 novembre 2014, 11:59:10
```

- Elle permet d'assigner la sortie d'une commande à une variable ou d'utiliser la sortie d'une commande comme un argument d'une autre commande.

Exemple 4:

```
% x=$(ls)
% echo $x
  ProgDebut   res
% rep=(pwd)  ;   echo $rep
  /home/etudiant/cours
%cd /
%cd $rep ;  pwd
  /home/etudiant/cours
```

III. Calcul numérique sur les entiers

1. Introduction

Il n'y a pas de notion de typage en shell, toutes les valeurs sont des chaînes de caractères.

En shell, toute expression ne contenant que des chiffres peut être utilisée pour du calcul arithmétique, sur des entiers, à l'aide des commandes « let, expr et (()) ».

Exemple1:

```
%val=10
```

- test1: on ne met pas d'espaces avant et après le symbole « + »

```
%res=$val+2
%echo $res      équivalente à   %echo  10+2
  10+2
```

- test2: si on met l'espace avant et/ou après le symbole « + », un message d'erreur est affiché. Par exemple on fait un espace avant et après le symbole « + ».

```
%res=$val + 2    équivalente à   %res=10 + 2
  + : not found.
```

2. La commande « expr »

Syntaxe :

```
expr expression
```

- La commande « expr » évalue l'expression arithmétique et affiche le résultat sur la sortie standard.

- L'expression concerne les opérations arithmétiques simples : + (addition), - (soustraction) , * (multiplication), /(division entière), % (modulo: le reste de la division entière)
- On peut effectuer des regroupements grâce à des parenthèses.
- Pour affecter le résultat de l'expression à une variable, la commande «expr» doit être mise entre les back-quotes: `expr expression`
- Code retour : si l'expression est valide le code retour est soit 0 ou 1. Si l'expression n'est pas valide alors le code retour vaut une autre valeur (en général 2).

Attention: Les membres de l'expression doivent être séparés par des espaces.

Exemple:

```
% expr 7 + 3
10
% a=3
%b=`expr $a + 1`
%echo $b
  4
% echo `expr $b / 3 `
  1
%b=`expr 6 % 4`
%echo  $b
  2
```

Attention :

Les caractères spéciaux « * », « (» et «) » doivent être précédés par le caractère « \ », afin d'éviter qu'ils soient interprétés par le shell avant de passer le contrôle à la commande « expr ».

Exemple:

- Calculer 2*3. Dans ce cas on doit précéder le caractère « * » par « \ »

```
%res=`expr  2 \* 3 ` ;  echo $res
6
%
```

- Calculer (res+a)*b. Dans ce cas on doit précéder les caractères spéciaux « * », « (» et «) » par « \ ».

```
%a=2; b=4
% d=`expr \( $res + $a \) \* $b`
% echo $d
  32
```

3. Evaluation arithmétique avec « (( )) »

Syntaxe:

$((expression))

Permet d'évaluer l'expression arithmétique « expression ». L'expression peut contenir des parenthèses sans le faire précédés par « \ ».

Exemple:

```
% a=12
%echo $(( a + 3 ))   ou  %echo $(( $a + 3 ))
  15
%b=$((a/5)) ; echo $b
  2
%c=$((a *b)) ; echo $c
  24
%d=$(((a + c)*b)) ; echo $d
  72
```

4. Evaluation arithmétique avec la commande « let »

Syntaxe :

let var =expresion

La commande « let » permet de faire l'évaluation suivie d'une affectation.

Exemple:

```
% let a="1+2" ; echo $a   ou   %let a=1+2 ; echo $a
  3
%a=8
```

```
%let var=(2+$a)*5  ou   %let var=(2+a)*5   ou  %let var="(2+$a)*5"
  50
%let c="${a}4 + 4" ;  %echo $c
  88
%let d='a+4'
%echo $d
  7
```

Important : Attention au sens de « " » et « ‘ »

IV. Les commandes d'entrée sorties

1. La commande « read »

Syntaxe :

read variables

La commande "read" permet de faire des lectures sur l'entrée standard (clavier). Les valeurs saisies sont séparées par des espaces. La fin de la saisie est détectée lorsqu'on tape sur la touche « Entrée ». Si la commande « read » n'a pas d'arguments (aucun nom de variable n'est spécifié), les valeurs lues sont mises dans la variable « REPLY ».

Exemple:

```
$ read
  Bonjour les Etudiants
$ echo $REPLY
   Bonjour les Etudiants
```

1er cas: Le nombre de valeurs saisies est égale au nombre d'arguments (de variables) de la commande « read ».

Exemple : On suppose que la commande « read » à 3 variables « var1, var2 et var3 » et qu'on saisi 3 valeurs « val1, val2 et val 3 »

```
%  read var1 var2 var3
  val1 val2 val 3
%echo $var1 ; echo $var2 ; echo $var3
  val1
```

val2

val3

- val1 est affectée à la variable var1
- val2 est affectée à la variable var2
- val3 est affectée à la variable var3

2ème cas: Le nombre de variables dans la commande « read » est plus grand que le nombre de valeurs saisies. Dans ce cas, les valeurs « NULL » (chaine vide) sont assignés, aux dernières variables (les variables les plus à droites).

Exemple: On suppose que la commande « read » à 3 variables « var1, var2 et var3 » et qu'on saisi 2 valeurs « val1, val2 »

```
% read var1 var2 var3
  val1 val2
%echo $var1 ; echo $var2 ; echo $var3
  val1
  val2
```

- val1 est affectée à la variable var1
- val2 est affectée à la variable var2
- NULL (chaine vide) est affecté à la variable var3

3ème cas: Le nombre de valeurs saisies est plus grand que le nombre de variables dans la commandes « read » alors le reste des valeurs saisies sont affectées à la dernière variable.

Exemple: On suppose que la commande « read » à 3 variables « var1, var2 et var3 » et qu'on saisi 5 valeurs « val1, val2, val3, val4 et val5 »

```
% read var1 var2 var3
  val1 val2 val3 val4 val5
%echo $var1 ; echo $var2 ; echo $var3
  val1
  val2
  val3 val4 val 5
```

- val1 est affectée à la variable var1
- val2 est affectée à la variable var2
- val3 val4 val5 est affectée à la variable var3

2. La commande « echo »

C'est une commande centrale du shell. Elle affiche, sur la sortie standard (écran), ses arguments.

Syntaxe :

echo arguments

- Les arguments peuvent être du texte (chaine) ou des variables, dans ce dernier cas, si la variable est précédé par « $ », la commande « echo » affiche sa valeur.
- Aux arguments, on peut rajouter des options (caractères) de formatage.

Options:

- « -n » : pas de passage automatique a la ligne.
- « -e » active l'interprétation des codes suivants qui doivent être écrits entre guillemets (doubles quotes).

  \c : comme -n

  \n : saut de ligne

  \t : tabulation

  \\ : caractère \

Exemples

```
% echo Bonjour
        Bonjour
%echo 5 + 8
        5 + 8
% echo HOME
        HOME
% echo $HOME
        /home/etudiant
 % echo 1 ; echo 2
        1
        2
% echo -n 1 ; echo 2
        12
% echo -e "1\t 2"   ou   % echo -e 1 "\t"  2
        1       2
```

V. Script shell

1. Introduction

Un script shell (ou un shell-script ou tout simplement un script) est un fichier qui contient un ensemble de commandes UNIX. A l'exécution, le fichier est interprété: les commandes sont exécutées automatiquement.

⇒ Le Script shell

- Est un programme qui utilise des commandes Unix.
- C'est un langage de programmation interprété
- Il permet l'automatisation des actions. Il est très utile pour automatiser des tâches répétitives.

2. Structure d'un script shell

La structure générale d'un script shell est la suivante:

```
#! Shell_utilisateur
code
```

- Sur la première ligne on donne le chemin absolu du shell utilisé.

Exemple:

```
#! /bin/bash
```

- Ensuite on écrit le code

Remarques 1:

- Une ligne commentaire est une ligne précédée par le caractère « # ». Les lignes de commentaire ne sont pas interprétées.
- La commande « exit » est utilisée pour terminer un script.
- A la fin du code, on peut insérer l'instruction « exit num ». L'entier « num » est utilisé comme code retour (code de sortie) du script. L'entier « num » est un nombre entre 0 et 255.
- Lorsqu'un script ne se termine pas avec « exit num» ou se termine avec « exit » sans paramètre, alors le code retour du script est le code retour de la dernière commande exécutée dans le script (avant **exit**).

Remarques 2:

- On a déjà vu qu'une séquence de commandes peut s'écrire en séparant les commandes avec « ; »

  commande1 ; commande 2 ;...

- Lorsque différentes commandes sont écrites sur des lignes différentes, alors elles sont exécutées l'une après l'autre.

  commande1
  commande2
  ….

  ⇔

  commande1 ; commande 2 ;...

Exemple: Ecrire un script shell qui affiche un message, ensuite liste les fichiers, avec leurs numéros d'inodes, du répertoire de connexion et enfin affiche la liste des processus.

```
# !/bin/bash
# le code:
echo "Bonjour !"
cd
ls –i
ps
exit 0
```

3. Exécution d'un script shell

Supposons que le nom de ce script shell est « prog_debut », pour l'exécuter, l'utilisateur doit avoir le droit d'accès en exécution: il doit avoir la permission « x ».

- Si le path contient le chemin du répertoire qui contient le script, alors pour l'exécuter, il suffit de taper le nom du ficher comme une commande.

Exemple:

```
% progdebut
  Bonjour !
  1182676  prog_debut     1182230 tache1.c
   PID          TTY          TIME         CMD
   2136         pts/1        00:00:00     bash
   2362         pts/1        00:00:00     prog_debut
   2364         pts/1        00:00:00     ps
%
```

- Si le path ne contient pas le chemin du répertoire qui contient le script, alors pour l'exécuter, il faut préciser son chemin.

  Exemple: Supposons que le script à exécuter est dans le répertoire courant et que le répertoire courant n'est pas dans le path, dans ce cas , il faut préciser que le script se trouve dans le répertoire courant comme suit:

```
%./prog_debut
   Bonjour !
  1182676  prog_debut      1182230 tache1.c
   PID   TTY          TIME         CMD
   2136  pts/1        00:00:00     bash
   2362  pts/1        00:00:00     prog_debut
   2364  pts/1        00:00:00     ps
%
```

4. Variables pour le passage d'arguments

4.1. Introduction

A un script shell, on peut lui passer, sur la ligne de commande, des arguments. Ces arguments (appelés aussi des paramètres positionnels) peuvent être exploités dans le programme lui-même comme des variables. D'autre paramètres spéciaux, sont prédéfinis et qui sont exploitables dans le script shell.

4.2. Les paramètres positionnels

Pour référencer les paramètres positionnels (les arguments passés sur la ligne de commande au script shell), on utilise les variables réservées 1,2,3,...9,10,11,... Ces variables peuvent être utilisées à l'aide des expressions $1,$2... (ou ${1},${2}...) où « $n » désigne le nième argument (exemple: $3 est le 3ième argument).

Attention: Dans certains cas, les accolades « {} » sont nécessaires:

Exemple1: considérons le script suivant dont le nom est « test »:

```
# !/bin/bash
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12
exit 0
```

Exécution

```
% ./test  un deux trois quatre cinq six sept huit neuf dix onze douze
  un deux trois quatre cinq six sept huit neuf un0 un1 un2
%
```

Exemple 2

```
# !/bin/bash
echo $1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ${12}
exit 0
```

Exécution

```
% ./test un deux trois quatre cinq six sept huit neuf dix onze douze
  un deux trois quatre cinq six sept huit neuf dix onze douze
%
```

4.3. Les paramètres spéciaux

Ce sont en fait des variables réservées. Certaines de ces variables sont liées aux arguments passés au script shell. Pour les utiliser, on les fait précédé par le caractère spéciale « $ ». Ces paramètres sont :

- La variable réservé « 0 »: $0 désigne le nom du script shell.
- Le paramètre « * » : $* revoie la liste de tous les arguments mais sous la forme d'un seul argument.
- Le paramètre « @ »: $@ renvoie la liste de tous les arguments mais sous forme d'arguments (les arguments son séparés par des espaces).
- Le paramètre « # » : $# renvoie le nombre d'arguments passés au script.
- Le paramètre « ? »: $? revoie le code retour de la dernière commande.
- Le paramètre « $ »: $$ renvoie le PID du processus qui exécute le script (le PID du sous-shell lancé par le shell en cour pour exécuter le script).
- Le paramètre « ! » : $! renvoi le PID du processus lancé pour exécuter la dernière commande lancée en arrière plan.

Exemple: Considérons le script suivants

```
# !/bin/bash
    echo "Bonjour !"
    echo " Le nom du script est : $0 "
    echo " Le nombre d'arguments est : $# "
    ls -i
    ps
    echo " Le PID du processus qui exécute le script est : $$ "
    echo " L'ensemble d'arguments  est : $* "
    echo " La liste de tous les arguments est : $@ "
    echo " Le premier argument est : $1 "
    echo " Le troisième argument est : $3 "
    exit 0
```

Exécution du script

```
% ./prog_debut un deux "trois quatre" cinq six
  Bonjour !
  Le nom du script est : ./prog_debut
  Le nombre d'arguments est : 5
  1182676  prog_debut     1182230 tache1.c
  PID          TTY          TIME         CMD
  1702         pts/0        00:00:00     bash
  3348         pts/0        00:00:00     prog_debut
  3350         pts/0        00:00:00     ps
  Le PID du processus qui exécute le script est : 3348
  L'ensemble d'arguments  est : un deux trois quatre cinq six
  La liste de tous les arguments est : un deux trois quatre cinq six
  Le premier argument est : un
  Le troisième argument est : trois quatre
%
```

Remarque:

Les arguments peuvent être séparés par des espaces ou des tabulations (il suffit d'un seul espace ou une seule tabulation).

N.B. Les espaces ou tabulations supplémentaires, délimitant les arguments du script seront éliminés.

Exemple:

```
% nom_prog   arg1 arg2          arg3          arg4
```

$* contient la liste des arguments séparés par un seul espace. Les espaces supplémentaires sont ignorés : arg1 arg2 arg3 arg4

VI. Les tests

1. La Commande « test »

La commande « test » est une commande externe au Shell. Elle est utilisée pour évaluer si une expression est vraie ou fausse. Elle est utilisée par d'autres commandes du shell: les branchements conditionnels ("if"), les boucles, … etc.

Syntaxe :

test expression
ou
[expression]

La commande « test » évalue l'expression spécifiée en argument et renvoie:

- 0 si le test est VRAI,
- une valeur différente de 0 si le test est faux.

Remarque: S'il n'y a pas d'expression, la commande renvoie 1 (false**).**

Attention: Les espaces avant et après les crochets « [» et «] » sont obligatoires.

2. Tests sur les fichiers

Syntaxe :

test option fichier
ou
[option fichier]

Les options usuelles sont

- test -a fiche ou [-a fiche]: vrai si le fichier « fiche » existe
- test -s fiche ou [-s fiche]: vrai si « fiche » est un fichier de taille non nulle.
- test -f fiche ou [-f fiche]: vrai si « fiche » est un fichier ordinaire.
- test -d fiche ou [-d fiche]: vrai si « fiche » existe et c'est un répertoire.
- test -L fiche ou [-L fiche] : vrai si « fiche » est un lien symbolique.
- test -r fiche ou [-r fiche]: vrai si l'utilisateur a le droit « r » sur le fichier « fiche » (vrai si le fichier existe et est autorisé en lecture).
- test -w fiche ou [-w fiche]: vrai si l'utilisateur a le droit « w » sur le fichier « fiche » (vrai si le fichier existe et est autorisé en écriture)
- test -x fiche ou [-x fiche] : vrai si l'utilisateur a le droit « x » sur le fichier « fiche » (vrai si le fichier existe et est autorisé en exécution).

3. Les tests sur les chaines de caractères

- test str1 = str2 ou [str1 = str2]: vrai si les deux chaines « str1 » et « str2 » sont identiques.
- test str1 != str2 ou [str1 != str2] vrai si les deux chaines « str1 » et « str2 » sont différentes.
- test str1 \< str2 ou [str1 \< str2] vrai si « str1 » est inférieure à « str2 ».
- test str1 \> str2 ou [str1 \> str2] vrai si « str1 » est supérieure à « str2 ».
- test -z str ou [-z str] vrai si « str » est nulle (non initialisée).
- test -n str ou[-n str] vrai si « str » est non nulle

équivalent au cas par défaut

test str ou [str] vrai si « str » est non nulle.

N.B.

- Il faut les espaces avant et après les crochets « [» et «] » et avant et après les signe de comparaisons.
- Si « str1 » ou « str2 » ou « str » sont des variables, alors il faut rajouter «$» pour accéder à leur contenu.

Remarque:

Il vaut mieux mettre les variables shell entre double quottes " ". Par exemple, considérons le test suivant :

[$X = oui]

Ou X est une variable initialisée (X est NULL), après évaluation, le shell exécute le test suivant :

[= oui]

qui est syntaxiquement incorrecte et par conséquent conduit à une erreur. Si on utilise les doubles quottes, on aura à exécuter ["$X" = oui] ce qui donne après évaluation du shell : ["" = oui] ce qui signifie à comparer oui avec la chaine vide.

4. Les tests numériques

Syntaxe :

test nombre relation nombre

ou

[nombre relation nombre]

Les relations sont :

- -lt (lower than): Strictement inférieur à
- -le (lower or equal): Inférieur ou égal à
- -gt (greater than): Strictement supérieur à
- -ge (greater or equal): Supérieur ou égal à
- -eq (equal): Egal à
- -ne (not equal): Différent de

Exemple d'utilisation: Soient n1 et n2 deux variables entières

[$n1 -eq $n2] : vrai si n1 est égal à n2

[$n1 -neq $n2] : vrai si n1 est différent de n2

[$n1 -lt $n2]: vrai si si n1 est inférieur à n2

[$n1 -gt $n2] : vrai si n1 est supérieur à n2

[$n1 -le $n2] :vrai si n1 est inférieur ou égal à n2

[$n1 -ge $n2] : vrai si n1 est supérieur ou égal à n2

5. Compositions logiques

ET logique: « && » ou « -a »

[expr1 -a expr2]

ou

test exp1 -a exp2 ou test exp1 && exp2

vrai si les deux expressions « expr1 » et « exp2 » sont vraies.

OU logique: « || » ou « -o »

[expr1 -o expr2]

ou

test exp1 -o exp2 ou test exp1 || exp2

vrai si au moins une expression est vraie.

La négation

test ! expr ou [! exp1] : vrai si l'expression est fausse (et vice-versa)

Exemple:

```
%val =53
% [ $val -gt 10  -a  $val -lt 100 ]
ou
% [ \($val -gt 10\) -a \($val -lt 100\) ]
% echo $?
      0
% [ $val -gt 100 -o $val -lt 0 ]
% echo $?
      1
% [ $val -lt 35 -a -a fiche]
% [ $val -lt 35 -o -x fiche]
```

VII. Structure de contrôle conditionnelle: « if »

1. La forme conditionnelle if…then

Il est possible d'exécuter une suite d'instruction sous une certaine condition.

```
if [ exp_cond ]                 |  if  test exp_cond
then                            |  then
        instructions            |          instructions
fi                              |  fi
                                |
if [ exp_cond ] ; then          |  if  test exp_cond ; then
        instructions            |          instructions
fi                              |  fi
```

« exp_cond » est une expression conditionnelle.

Si « exp_cond » est vrai (son code retour est égale à 0) alors les instructions qui se trouvent après le mot clé « **then** » sont exécutées.

Remarque:

- Les espaces dans if [test] then sont obligatoires.
- Quand « if » et « then » sont sur la même ligne, alors un point-virgule doit les séparer.

Exemple 1:

- Afficher le contenu d'un fichier, passé en argument, s'il existe.

```
#!/bin/bash
if [ -f $1 ]   ou   if test -f $1
then
  cat $1
fi
```

- Créer le répertoire « rep » s'il n'existe pas.

```
#!/bin/bash
if [ ! -d rep ] ; then
  mkdir rep
fi
```

Exemple 2: Script qui teste si un nombre, passé en argument, est positif et inférieur à 100.

```
#!/bin/bash
if [ \( $1 -gt 0 \) -a \( $1 -lt 100 \) ]
then
  echo " $1 est compris entre 0 et 100 "
fi
```

On peut ne pas mettre les parenthèses

```
#!/bin/bash
if [ $1 -gt 0 -a $1 -lt 100 ]
then
  echo " $1 est compris entre 0 et 100 "
fi
```

Remarque: La forme « if … then » peut être écrite avec les opérateurs logiques du shell : « && » et « || »

- **ET logique « && »:**

```
if cmd1
then
     cmd2
fi
⇔
cmd1 && cmd2
```

Si l'exécution de la commande « cmd1 » est bien passé alors on exécute commande « cmd2 ».

Exemple: gcc toto.c -o toto && ./toto

- **OU logique « || »:**

```
if ! cmd1
then
    cmd2
fi
```

⇔

```
cmd1 || cmd2
```

Si la commande « cmd1 » a échoué alors on exécute la commande « cmd2 ». Exemple :

```
cat toto || touch toto
```

2. La forme if…then .. else

Syntaxe:

```
if [ exp_cond ]          ou          if test exp_cond
    then
      instructions1
    else
    instructions2
fi
```

Dans cette syntaxe, si «exp_cond » est vrai (retourne la valeur 0) alors les instructions qui se trouvent après le mot clé « **then** » sont exécutées sinon ce sont les instructions qui se trouvent après le mots clé « **else** » qui sont exécutées.

3. Branchement conditionnel : if-then-elif-else-fi

```
if [ exp_cond1 ]
then
   instructions1
elif [ exp_cond2 ]
then
  instructions2
else
  instructions3
fi
```

Dans cette syntaxe, si « exp_cond1 » est vrai (retourne la valeur 0) alors la liste d'instructions « instructions1 » qui est exécutée, sinon, si «exp_cond2 » est vrai (retourne la valeur 0) alors c'est la liste d'instructions « instructions2 » qui est exécutée; sinon exécutions de la liste d'instructions « instructions3 ».

Exemple:

Supposons qu'on a seulement trois types de fichiers dans le répertoire courant : répertoire , lien symbolique ou fichier ordinaire . Le script suivant affiche le type du fichier passé en argument.

```
if [ -d toto ]
then
  echo "toto est un répertoire"
elif [ -h toto ]
then
  echo "toto est un lien symbolique"
else
  echo " toto est un fichier ordinaire" faut pousser l'investigation plus loin"
fi
```

VIII. Structure de contrôle conditionnelle : « case-esac »

Analogue au « case » du langage « C ». Elle est introduite par le mot-clé « case », et termine par la mot-clé « esac ».

Syntaxe :

```
case val in
  cas_1) instructions_1 ;;
  cas_2) instructions_2 ;;
  ...
  cas_n) instructions_n ;;
esac
```

- Tout d'abord « val » est évalué.
- Le résultat de « val » est comparé (en tant que chaine de caractères) avec chaque cas « cas_i ».
- Les instructions associées au premier cas qui correspond à «val» seront exécutées, puis on passe le contrôle à la première instruction après « esac ».

Règles pour les cas :

- Ils peuvent contenir des caractères génériques du shell : * ? [ ]
  * : remplace n'importe quelle séquence de caractères,
  ?: remplace un unique caractère,
  [ ] : un caractère parmi ceux définis entre crochets.
- Les cas peuvent être juxtaposés avec le symbole « | » qui signifie « OU ».

Remarque:

- Les instructions de chaque « cas_i » doivent se terminer par deux point-virgule accolés.
- Le cas « default » est obtenu en prenant « * » pour la cas «cas_n». Il doit être placé à la fin de la structure de contrôle «case» (le dernier cas de « case »).
- Le code de retour de la structure « case .. esac » est celui de la dernière commande exécutée de « instructions_i ».

Exemple

Le script suivant lit au clavier une chaine. Suivant la valeur de cette chaine on lance la commande « ls » avec ou sans options.

```
#!/bin/bash
echo " Voulez-vous lancer ls -l ou ls  ? "
read rep
case $rep in
        [Yy] | [oO] |  oui |  yes )  echo " Lancement de la commande ls -l "
                                ls -i ;;
        [Nn]*)    echo " Lancement de la commande ls "
                        ls ;;
        *) echo " Votre réponse n'est pas juste" ;;
esac
```

IX. Structures itératives

Elles permettent de répéter l'exécution d'un bloc d'instructions (un bloc de commandes) . Toutes les commandes à exécuter dans une boucle se placent entre les commandes « do » et « done ».

1. Boucle for

La boucle « for » n'est pas basée sur l'incrémentation d'une valeur mais sur une variable parcourant une liste dont les éléments sont séparés par des espaces.

Syntaxe:

```
for var in list
do
    list_instructions
done
```

```
for var in list ; do
        list_instructions
done
```

- La liste « list » représente un ensemble d'éléments séparés par des espaces.
- La variable « var » prend successivement les éléments de la liste « list ».
- Pour chaque élément, le bloc d'instructions « list-instructions » sont exécutées.
- Si la liste « list » contient des substitutions, elles sont traitées par le shell avant de passer le contrôle à « for ».

1.1. Cas d'une liste d'éléments donnés explicitement :

Exemple1: Considérons le script suivant « test_for1 »

```
#!/bin/bash "
for  mois  in  janvier février mars avril
do
  echo $mois
done
% ./test_for1
 janvier
février
mars
avril
%
```

Exemple2 : Considérons le script suivant « test_for2 »

```
#!/bin/bash
var=/home/etudiant/TP
for i in tmp  $var
do
    echo "$i"
done
```

```
%./test_for3
        tmp
        /home/etudiant/TP
%
```

Remarque :

Toute chaîne de caractères comprises entre les caractères « " » ou « ’ » comptent pour une seule itération pour la boucle « for ».

Exemple 3: Considérons le script suivant « test_for3 »

```
        #!/bin/bash
        for  mois in  janvier " février mars " avril
        do
          echo $mois
        done
% ./test_for3
         janvier
        février mars
        avril
%
```

Comporte 3 itérations au lieu de 4 comparé au script « test_for1 ».

Exemple 4: Considérons le script « test_for4 » suivant qui considère une liste d’entiers.

```
        # !/bin/bash
        for  I  in 1 2 9 12 ; do
          echo "$i² = $((i * i))"
        done
% ./test_for4
1² = 1
2² = 4
9² = 81
12² = 144
```

1.2. Cas d'une liste implicite

Si aucune liste n'est précisée à la boucle « for », alors c'est la liste des paramètres positionnels qui sera considéré.

Exemple: Considérons le script suivant « test_for »

```
#!/bin/bash "
for arg
do
        echo "$arg"
done
%./test_for  un "deux trois"  quatre
un
deux trois
quatre
%
```

1.3. Cas d'une liste explicite de paramètres positionnels

Exemple1: Considérons le script suivant « test_for1 »

```
#!/bin/bash "
for arg in "$@"
do
        echo "$arg"
done
%./test_for1  un "deux trois"  quatre
un
deux trois
quatre
%
```

Exemple2: Considérons le script suivant « test_for2 »

```
#!/bin/bash "
for arg in "$*"
do
        echo "$arg"
done
```

```
%./test_for2  un "deux trois"  quatre
un  deux  trois  quatre
%
```

1.4. Cas d'une liste de noms de fichiers définis à partir des caractères génériques de noms.

Exemple: Considérons le script « test_for » suivant:

```
#!/bin/bash
        for x in *.c
        do
          echo " $x est un fichier .c "
        done
%./test_for
test1.c  est un fichier .c
tache2.c est un fichier .c
calcul.c est un fichier .c
%
```

1.5. Cas d'une liste obtenue après substitution de commande

Exemple: Considérons le script « test_for» suivant:

```
#!/bin/bash
        echo " la date est : "
        for x in `date`
        do
          echo "$x "
        done
```

```
#!/bin/bash
 echo " la date est : "
for x in  $(date)
        do
         echo "$x "
done
```

```
% ./test_for
mardi
25
novembre
2014,
11:59:10
%
```

2. Boucle while-do-done

La commande « while » permet de réaliser la boucle conditionnelle «tant que».

Syntaxe:

```
while condition
do
  liste_instructions
done
```

ou

```
while
  bloc d'instructions formant la condition
do
  liste_instructions
done
```

- Dans la première forme : tant que la condition retourne vrai (retourne 0), le bloc « liste_instructions » est exécuté. On sort de la boucle si la condition retourne false (une valeur différente de 0).
- Dans la deuxième forme: le bloc «liste_instructions » est exécuté tant que la dernière commande du bloc d'instructions formant la condition est "vrai" (statut égal à zéro). On sort de la boucle si le statut est différent de 0.

Exemple1: considérons le script « test_while1» suivant:

```
#!/bin/bash
let i=0
while test $i -lt 3
do
    echo $i
    let i=i+1
done
% ./test_while
    0
    1
    2
%
```

Exemple 2: Considérons le script « test_while2» suivant:

```
#!/bin/bash
while
  echo " Entrer un entier "
  read n
  [ $n -lt  0 ]
do
  echo " Erreur: entrer une valeur supérieure à 0 "
done
echo " La valeur saisie est : $n"
```

3. Contrôle d'exécution

- **La commande « break »:** Permet de sortir de la boucle en cours et passe à la première commande qui suit « done ».
- **La commande « continue »:** permet d'interrompre l'itération en cours et passer à l'itération suivante.
- **La commande « exit »:** Provoque l'arrêt du programme shell.

Syntaxe:

```
exit [n]
```

- Si n est spécifié, le code retour du script shell est positionné à «n».
- Si n n'est pas spécifié, alors le code de retour du script shell sera positionné à la valeur du code retour de la dernière commande exécutée juste avant "exit".

Exemple1: Exemple avec la commande « break »

```
#!/bin/bash
while true
do
    echo " Entrer un entier "
    read n
    if  [ $n -lt  0 ] ;  then
       break
    fi
done
echo " La dernière valeur saisie est : $n"
```

Exemple2: Exemple avec la commande « continue »

```
#!/bin/bash
        let i=0
        while test $i -lt 6
        do
          let i=i+1
          if test $((i%2)) -eq 0
          then
            continue
          fi
          echo $i
        done
% testcont
        1
        3
        5
```

X. Les fonctions

Dans un programme script, on peu regrouper un ensemble d'instructions dans une fonction qui sera identifiée par un nom. La fonction est déclarée comme suit :

1. Déclaration de la fonction

```
        nom_fct ()              ou              function nom_fct ()
        {
          #corps de la fonction
          liste-commandes
        }
```

- « nom_fct » désigne le nom de la fonction.
- « liste-commandes » définit le corps de la fonction.
- Le corps de la fonction peut contenir la commande « return » qui permet de mettre fin à l'exécution de la fonction et retourner le code retour de la fonction. Le programme appelant peut récupérer le statut de la fonction à l'aide de la variable "?" ($?).

2. Syntaxe de la commande « return »

return [n]

- Si le paramètre "n" est précisé, il correspond au "statut" de la fonction (la valeur du code retour de la fonction). Il est transmis au programme ayant appelé cette fonction. Le nombre « n » est compris entre 0 et 255. Par convention du shell, une valeur égale à 0 correspond à vrai et une valeur différente de 0 pour faux.
- Si le paramètre « n » n'est pas spécifié ou s'il n'y a pas la commande « return » alors le code de retour de la fonction est celui de la dernière commande exécutée dans la fonction.

3. Utilisation de la fonction

- Pour appeler une fonction (exécuter l'ensemble des instructions définies dans la fonction), il suffit de donner son nom comme une commande Unix.
- On peut passer à la fonction des arguments (ou paramètres) lors de l'appel de la fonction. Ces paramètres peuvent être référencés dans le corps de la fonction de la même façon que les arguments d'un script shell, c'est-à-dire en utilisant les variables $1, $2,... $@, $#

Remarque:

- Dans le script shell, ces variables font références aux paramètres positionnels (paramètre passé au script shell).
- Dans le corps de la fonction, ces variables font références aux avaleurs passées à la fonction.
- La valeur "$0" référence toujours (même dans le corps de la fonction) le nom du script.

Exemple d'appel d'une fonction:

nom_fct param1 param2 ….

où « parm1 », « param2 », désignent les paramètres (arguments) de la fonction.

Exemple

La fonction « pair_impair » dans le script suivant teste la parité des nombres passés en argument au script shell.

```
#!/bin/bash
# Définition de la fonction qui test la parité d'un nombre passé en argument à la fonction
pair_impair()
{
        if test `expr $1 % 2` -eq 0  ;  then
           echo "$1 est pair"
        else
          echo "$1 est impair"
        fi
}
# Test de la parité de tous les nombres passés au script shell
for nb in "$@"
do
        # Appel de la fonction pair-impair pour tester la parité
        pair_impair $nb
done
```

Remarque

- Une définition de fonction peut se trouver en tout point d'un programme shell.
- Une fonction doit être déclarée avant de l'utiliser.
- L'appel d'une fonction n'engendre pas la création d'un sous-shell pour l'exécuter. Elle est exécutée par le processus shell dans lequel elle a été définie.
- Une fonction n'est connu que dans le processus dans lequel elle a été définie.

  ⇒ une fonction déclarée dans un script n'est pas connue dans les processus engendrés par les script shell.
- La commande «**unset -f nomfct ...** » permet de rendre une fonction indéfinie , c'est-à-dire on ne peut plus l'utiliser après l'exécution de cette instruction.

Attention:

Puisque une fonction s'exécute dans le shell courant, alors tout appel à la commande "exit" dans la fonction entraine la fin du script shell.

4. Portée des variables

Puisque l'appel d'une fonction n'engendre pas la création d'un sous-shell pour l'exécuter, alors par défaut, une variable définie dans le script shell ou à l'intérieur d'une fonction est globale; c'est-à-dire qu'on peut l'utiliser et modifier dans les autres fonctions du programme shell. Pour définir une variable locale, on utilise le mot clé « local » .

Exemple : Dans le script suivant « test_local », on définit une fonction qui modifie deux variables : une « globale » et une « locale ».

```
#!/bin/bash
# Fonction qui modifie deux variables
modif()
{
        i=5    # i est globale
        local j=8           # j est locale
        j=12
        echo « Les valeurs de i et j dans la fonction: i=$i; j=$j"
}
# Programme principal
        i=0; j=0 # Initialisation de "i" et "j"
        echo « Avant appel :   i=$i; j=$j"
        modif      # Appel de la fonction
        echo « Après appel:    i=$i; j=$j"
```

```
%./test_local
Avant appel:  i=0   j=0
Les valeurs de i et j dans la fonction:  i=5    j=12
Après appel:  i=5   j=0
```

Remarque

La définition d'une fonction peut avoir plusieurs formes

```
nom_fct  {
        suite_de_commandes
}
function nom_fct () {
        suite_de_commandes
}
nom_fct ()
{ suite_de_commandes ;}
```

XI. Notion de tableaux

Sous linux, on peut utiliser les tableaux monodimensionnels. Les indices des tableaux commencent à 0.

- On peut faire une déclaration et initialisation de plusieurs éléments du tableau en même temps:

  Exemple:

  annee=(" janvier" " Février")

- On peut aussi faire la déclaration et l'initialisation individuelles

  Exemple:

  annee[0]=" Janvier"

  annee[1]=" Février"

- La taille est dynamique.

  Exemple:

  initialement on déclare et on initialise le tableau « annee »

  annee=(" Janvier" " Février")

  Ensuite on peut compléter avec d'autre éléments

  anne[2]=" Mars"

  anne[3]=" Avril"

- Chaque élément du tableau est considéré comme une variable. Pour accéder au contenu d'un élément on utilise l'indice de l'élément dans le tableau est la caractère « $ »

  Exemple:

  echo ${tab[0]} affiche le premier enregistrement du tableau « tab »

  echo ${tab[1]} affiche le deuxième enregistrement du tableau "tab"

  Attention:

  « echo ${tab} » identique à « echo ${tab[0]} »

- Le nombre d'éléments du tableau est désigné par :

  ${#tab[*]}

- L'ensemble d'enregistrements du tableau sont désignés par:

  tab[@] ou tab[*]

  Exemple : afficher tous les enregistrements du tableau « tab »

  ```
  echo ${tab[@]}
  ou
  echo ${tab[*]}
  ```

- On peut accéder aux éléments du tableau élément par élément

  Exemple:

  ```
  for (v in ${tab[*]}      ou      for (v in ${tab[@]}
  do
          echo ${v}
          #traitement sur « v »
  done
  ```

- L'ensemble des indices du tableau sont désignés par «${!tab[*]}».

  Exemple: afficher tous les éléments du tableau « tab » en uitilisant les indices.

  ```
  for i in ${!tab[@]}
  do
          echo ${tab[i]}      ou    echo ${tab[$i]}
  done
  ```

- Le nombre d'enregistrements dans le tableau est désigné par:

  « ${#tab[*]} » ou « ${#tab[@]} »

- La longueur du (i+1)ème élément du tableau (longueur de la chaine de caractère) est désignée par:

  « ${#tab[i]} »

  Exemple:

  « ${#tab[2]} » désigne la longueur du troisième élément du tableau « tab »

XII. Complément de commandes

1. La commande « tr »

La commande « tr » permet de remplacer une chaine par une autre.

Syntaxe:

tr [options] chaine1 chaine2 < fiche

La commande "tr" va remplacer « chaine2 » dans le fichier « fiche » par «chaîne1» et affiche le résultat sur la sortie standard.

N.B. le fichier « fiche » ne sera pas modifié avec la commande précédente.

Exemple :

% tr ":" " " < /etc/passwd ou % tr ":" " " < /etc/passwd

La commande remplace « : » dans le fichier « /etc/passwd » par un espace («' '»), le résultat s'affiche sur la sortie standard.

Les options :

- « -d » : efface le caractère spécifié.
- « -s » : si le caractère spécifié se répète plusieurs fois de suite, il est réduit à une seule unité.

2. Commande « paste »

La commande « paste » permet de rassembler les fichiers ligne par ligne et ensuite affiche le résultat sur la sortie standard. Les lignes sont regroupées l'une à coté de l'autre séparées par une tabulation (ou espace) et terminées par un retour chariot (return).

Syntaxe:

paste [option] fiche1 fiche2

Option:

- L'option « -d car»: indique que les lignes regroupées sont séparées par le caractère « car »

3. Commande « join »

Cette commande permet de fusionner les lignes de deux fichiers ayant un champ identique (appelé champ de fusion). Les deux fichiers doivent être triés par ordre croissant suivant les champs utilisés pour la fusion.

Syntaxe:

join [option] fiche1 fiche2

- La commande « join » affiche sur la sortie standard les lignes fusionnées. Une ligne affichée correspond à la fusion d'une ligne du fichier «fiche1» et d'une ligne du fichier «fiche2» qui ont les champs de fusion identiques.
- Par défaut la fusion s'effectue selon le premier champ où les champs sont séparés par des espaces. Les espaces en début de ligne sont ignorés, de même, les champs en sortie sont séparés par des espaces.

Quelques options

- « -j1 champ » ou « -1 champ»: désigne le numéro de champs de la fusion du fichier 1.
- « -j2 champ » ou « -2 champ»: désigne le numéro de champs de la fusion du fichier 2.
- « -j champ » : équivalent à « -1 champ -2 champ ».
- « -t car »: le caractère « car » désigne le séparateur de champs pour les fichiers d'entrée et pour la sortie.

Exemple:

%join fiche1 fiche2 -t: -j1 3 -j2 1 ou %join fiche1 fiche2 -t: -1 3 -2 1

Permet de faire la fusion des fichiers « fiche1 » et « fiche2 » où les champs de la fusion sont le champ 3 pour le fichier « fiche1 et le champ1 pour le fichier « fiche2 ». Le séparateur de champs est le caractère « : »

4. Commande « uniq »:

La commande « uniq » permet d'éliminer les lignes identiques dans un fichier trié.

Syntaxe:

uniq [option] fichier_entrée fichier_sortie

- Par défaut, la commande « uniq » affiche sur la sortie standard les lignes uniques d'un fichier trié. Elle n'affiche qu'un seul exemplaire de chaque ligne dupliquée.
- Si le fichier de sortie est mentionné, la sortie de la commande est redirigée dans ce fichier.

Quelques options

- « -u »: Seulement les lignes non dupliqués (lignes uniques) qui sont affichées.
- « -d »: n'affiche que les lignes dupliquées.
- « -c »: pour chaque ligne affichée, son nombre d'occurrences est affiché.

5. La commande « touch »

La commande « touch » modifie la date de chaque fichier indiqué.

Syntaxe:

touch [option] fiche...

Par défaut:

- Si le fichier n'existe pas alors il sera créé avec un contenu vide.
- Si le fichier existe, alors la date et l'heure du fichier seront remplacées par la date et l'heure actuelles.

Options:

- L'option « -c »: ne crée pas le fichier « fiche » s'il n'existe pas.
- L'option « -r fic »: remplace la date et l'heure du fichier « fiche » par celle du fichier « fic ».
- L'option « -t date »: remplace la date du fichier « fiche » par celle qui est indiquée. Le format de la date est :
  - MMJJhhmm[.ss] où « MM » désigne le mois, « JJ » désigne le jour, « hh » désigne l'heure et « mm » désigne les minutes. Evetuellement on peut préciser les secondes.
  - SSAAMMJJhhmm[.ss] où « SS » désigne le siècle et « AA » désigne l'année.

Exemple:

%touch -r test1 test2

Remplace la date du fichier « test2 » par celle du fichier « test1 »

%touch -t 09151456 test

La nouvelle date du fichier « test » est: 15 Septembre 2014 à 14h56

%touch -t 200509151456 test

La nouvelle date du fichier « test » est: 15 Septembre 2005 à 14h56

6. La commande « shift »

Cette commande permet le décalage des paramètres

Syntaxe :

shift [n]

La commande "shift" consiste à décaler tous les arguments de "n" positions vers la gauche, et décrémenter de « n » la variable "#" (nombre de paramètres postionnels).

- Si « n » n'est pas précisé, la commande « shift » décale tous les paramètres d'une position en supprimant le premier : $2 devient $1, $3 devient $2 et ainsi de suite

  ⇒ par défaut, "n" vaut 1.
- Si « n » est précisé, la commande « shift » effectue un décalage de « n » positions vers la gauche en supprimant les « n » premiers paramètres: Par exemple avec « shift 4 », $5 devient $1, $6 devient $2, ...
- Les variables « $# », « $* » et « $@ » sont redéfinis en conséquence, par exemple un « shift » de « n » positions, décrémente « $# » de « n » et « $* » et « $@ » ne contiennent que les paramètres à partir du (n+1) ème.

Exemple: considérons le script « test_param » suivant:

```
# !/bin/bash
echo "arg1= $1, arg2= $2, arg3 = $3, arg4 = $4, arg5 = $5, total : $#"
shift
echo “Après shift: arg1=$1, arg2=$2, arg3=$3, arg4=$4, arg5=$5; total: $#"
shift 2
echo “Après shift 2: arg1= $1, arg2 = $2, arg3 = $3, arg4 = $4, arg5 = $5; total : $#"
exit 0
```

Exécution:

```
%./ test_param un deux trois quatre cinq
arg1= un, arg2= deux, arg3 = trois, arg4 = quate, arg5 = cinq, total : 5
Après shift: arg1= deux, arg2= trois, arg3= quatre, arg4= cinq, arg5= ;total: 4
Après shift 2: arg1=quatre, arg2= cinq, arg3= , arg4= , arg5= ; total : 2
%
```

XIII. Choix interactif: « select »

- Cette commande permet d'effectuer un choix interactif parmi une liste proposée en affichant un menu simple.
- Le choix se fait par numéro.
- La saisie des choix s'effectue au clavier.
- Après chaque saisie, une boucle s'effectue et le menu s'affiche à nouveau jusqu'à ce qu'on rencontre un « break ».

Syntaxe

```
select var in liste
do
  list_instructions
done
```

Si « in liste » n'est pas précisé, se sont les paramètres positionnels qui sont utilisés et affichés.

Principe de fonctionnement

1. Tout d'abords, la liste des choix est affichée sur la sortie standard.
2. On saisi un choix. Si le choix est valide, la valeur saisie est affectée à la variable « var ».
3. Le corps de la boucle est exécuté.
4. On passe de nouveau à l'étape « 1 » et le menu s'affiche à nouveau, jusqu'a ce que l'instruction « break » soit rencontrée.

N.B. La variable « PS3 » définit l'invite de saisie pour un «select». Pour personnaliser l'invite de saisie pour « selecte », on modifie le contenu de cette variable.

Exemple 1: Considérons le script « test_select1 » suivant:

```
#!/bin/bash
echo " Que voulez vous faire ? "
select rep in o n q
do
  if [ "$rep" = "q" ]
  then
        break
  fi
  echo "Votre choix est : $rep"
done
echo « Fin"
```

```
% ./test_select1
Que voulez vous faire ?
        1) o
        2) n
        3) q
        #? 1
        #? 3
        Le choix est: q
        Fin
%
```

Exemple 2: Considérons le script « test_select2 » suivant:

```
#!/bin/bash
echo "  Que voulez vous faire ? "
select v in pwd ps Quitter
do
  case $v in
  "pwd") pwd ;;
   " ps") ps ;;
  "Quitter") break ;;
  *) echo "Taper 1 2 ou 3 "
  esac
done
echo " Fin "
```

```
%./test_select2
Que voulez vous faire ?
        1) pwd
        2) ps
        3) Quitter
        #? :1
          /home/etudiant/cours
        #? :bonjour
          Taper 1 2 ou 3
        #? :2
        Fin
%
```

Exemple 3: Considérons le script « test_select3 » dans lequel on modifie l'invite de saisie (on modifie le contenu de la variable « PS3 »).

```
#!/bin/bash
PS3="Votre choix:"
select v in pwd  ps  Quitter
do
  case $v in
   " pwd") pwd ;;
   " ps") ps ;;
  "Quitter") break ;;
  *) echo "Taper 1 2 ou 3 "
  esac
done
echo " Fin "
```

```
%./test_select3
 1) pwd
 2) ps
 3) Quitter
 Votre choix: 1
  /home/pascal
 Votre choix: 7
  Taper 1  2 ou 3
 Votre choix: 2
Fin
%
```