Introduction au Système Unix/Linux

Document de cours

Prof. Hicham GIBET TANI
(gibet.tani.hicham@gmail.com)

Linux

Licence – Filière Sciences Mathématiques Informatiques (SMI)

Module – Programmation Système

Contenu

I. Unix vs Linux : 6
1. Définition Unix/Linux : 6
2. Histoire : 6
3. Le coût Unix/Linux : 7
II. Introduction à Linux : 8
4. Système d'exploitation linux : 8
5. Le Kernel Linux (noyau) : 8
6. Exploiter le système Linux : 9
7. Structure des fichiers Linux : 10
8. Types de systèmes de fichiers : 10
9. Utilitaires Linux : 10
III. Exploitation de Linux : 12
1. Authentification : 12
1.1. Nom d'utilisateur : 12
1.2. Mot de passe utilisateur : 12
1.3. Connexion réussie : 12
1.4. Déconnexion : 12
1.5. Changer le mot de passe utilisateur : 12
2. Les commandes Linux : 13
2.1. Format d'une commande : 13
2.2. Les options d'une commande : 13
2.3. Les arguments d'une commande : 13
2.4. Exemple : 13
2.5. Exceptions : 13
3. Les commandes date et cal : 14
4. Les commandes clear et echo : 14
5. Les commandes who et finger : 14
6. Envoyer un mail : 14
7. Recevoir un mail : 15
8. Gestion de messagerie électronique : 15
9. Les commandes write et wall : 16
10. La commande man : 16

IV. Fichiers et Répertoires : 17
1. Fichiers : 17
2. Répertoires : 17
2.1. Contenu des répertoires : 17
2.2. L'accès des utilisateurs aux fichiers : 17
2.3. i-node: 18
3. Système de fichier (File System) : 18
Exemples : 18
4. Les chemins d'accès aux fichiers : 19
Exemples : 19
5. Utilisation de la commande pwd : 19
Exemple : 19
6. Utilisation de la commande ls : 19
Exemple : 20
Affichage détaillé de la commande ls : 20
7. Utilisation de la commande cd : 20
8. Création et suppression d'un répertoire : 20
Exemple : 20
Exemple : 20
9. Création d'un fichier : 21
Exemple : 21
V. Utilisation des fichiers : 22
1. Copie : 22
Exemple : 22
2. Déplacer et renommer : 22
Exemple : 22
3. Contenu d'un fichier : 22
4. La commande « wc » : 23
5. Liaison de fichiers : 23
Exemple : 23
Lien symbolique : 23
Exemple : 23
6. Supprimer un fichier : 23

Exemple : 24
VI. Les permissions des fichiers : 25
1. Liste de fichier : 25
Exemple : 25
2. Les permissions : 25
Fichier : 25
Répertoire : 26
3. Modifications des permissions : (Notation symbolique) 26
Exemple : 26
4. Modifications des permissions : (Notation octal) 26
Exemple : 27
5. Les permissions par défaut : 27
6. Les permissions requises : 27
VII. L'éditeur de texte vi : 28
1. Démarrage de « vi » : 28
2. Ajouter du texte : 28
3. Arrêter l'éditeur « vi » : 28
4. Mouvement du curseur : 29
5. Supprimer du texte : 29
6. Rechercher du texte : 29
7. Remplacer du texte : 29
8. Couper, Copier et coller : 30
VIII. Les bases du SHELL : 31
1. Substitution des noms de fichiers : 31
1.1. Les caractères ? Et * : 31
1.2. Les listes d'inclusions : 31
1.3. Redirection de commande : 32
2. Pipe : 33
3. Filtre : 34
4. La commande « tee » : 34
5. Regroupement de commandes : 35
6. Continuation de ligne : 35
IX. Les variables SHELL : 36

1. Lister les paramètres des variables : 36
2. Configuration d'une variable : 36
4. Substitution des commandes : 37
5. Les méta-caractères : 37
5.1. Apostrophe : 37
5.2. Guillemet : 38
5.3. Backslash : 38
X. Les processus : 39
1. Le processus d'environnement : 39
2. Les relations processus : 40
3. Les variables et processus : 40
Activité : 40
4. Script SHELL : 41
Invoquer un script SHELL : 41
XI. Contrôle de processus : 43
1. Processus au premier plan : 43
2. Processus en arrière-plan : 43
3. Interrompre un processus : 43
1.1 Processus au premier plan : 43
1.2 Processus en arrière-plan : 43
1.3 La commande « kill » : 43
1.4 Les signaux de la commande kill: 44
4. Exécution des processus long : 44

Objectifs du Module

Ce module a pour objectif la présentation des fonctionnalités et mécanismes internes d'un système d'exploitation multitâches, au travers du cas du système UNIX/Linux. Les connaissances de cet enseignement permettront à l'étudiant de développer des applications, écrites en langage de haut niveau, faisant directement appel au noyau du système, et notamment des applications client-serveur susceptibles de fonctionner à travers un réseau.

- Introduction au système Unix/Linux
- Gestion du système de fichiers
- Gestion de la mémoire
- Gestion des processus
- Gestion de threads sous Linux
- Communication interprocessus
- Programmation réseau synchrone et asynchrone

Les Références

- Linux System Programming (Talking Directly To The Kernel And C Library), 2nd Edition. Robert Love. O'REILLY, May, 2013.
- Programmation système en C sous Linux, 2ème édition. Christophe Blaess. EYROLLES, 2005.
- Cours Programmation Système. Pr. Mohammed ACHKARI BEGDOURI, 2019.
- TCP/IP Sockets in C – Practical Guide for Programmers. Micheal J. Donahoo, Kenneth L. Calvert. The Morgan Kaufmann Practical Guides Series, 2001.
- https://www.geeksforgeeks.org/
- https://www.cse.cuhk.edu.hk/en/

I. Unix vs Linux :

1. Définition Unix/Linux :

Linux :

Les systèmes d'exploitation Linux sont des systèmes avec une source ouverte dont vous pouvez l'utiliser d'une manière libre. Ces systèmes sont largement utilisés comme plateforme pour les ordinateurs, les logiciels, le développement des jeux, les tablettes PC, les mainframes, etc.

https://www.kernel.org/

Unix :

UNIX représente les systèmes d'exploitation classiquement utilisé dans des serveurs et postes de travail **spécifique**.

2. Histoire :

Le développement des systèmes d'exploitation UNIX a commencé vers les années 60s (1960s) comme un projet collaboratif entre:

- MIT (Massachusetts Institute of Technology)
- AT&T (fournisseur de services téléphoniques)
- Bell Labs (société de recherche et développement scientifique)
- General Electric (Un groupe industriel mondial)

Le projet a mis en œuvre le système d'exploitation MULTICS conçus pour l'ordinateur GE-645.

- Le système d'exploitation a donné de très mauvais résultat et la collaboration a été rompue.
- Un des développeurs de « Bell Labs » (Ken Thompson) a continué le travail sur le projet et son travail a été couronné par la création de la première version des systèmes UNIX vers les années 1970s.
- En 1985, (Richard Stallman) a créé la fondation « Free Software Foundation » et la licence GNU General Public License (GNU GPL), afin de diffuser les logiciels librement. La problématique était l'exigence d'un système d'exploitation pour exécuté ces logiciels => La création du système GNU (GNU's Not Unix)
- En 1991, (Linus Torvalds) a commencé le dévelopment de « MINIX », un système d'exploitation similaire à UNIX, dont son code était librement disponible sous le projet GNU GPL. Ensuite, il a développé le premier noyau LINUX et qui était publié le 17 septembre 1991, pour les ordinateurs Intel.

3. Le coût Unix/Linux :

	Linux	Unix
Coût	Linux peut être distribué et téléchargé librement. Cependant, il existe des versions payantes pour Linux.	Les différentes versions d'Unix ont des structures de coûts selon les fournisseurs.
Développement et distribution	Linux est développé par la communauté Open Source, c'est-à-dire par le partage et la collaboration de code et de fonctionnalités à travers des forums, etc. (à l'exception des développements spécifiques sur les versions payantes)	Les systèmes Unix sont divisés en diverses versions, principalement développées par les différents fournisseurs commerciaux et les organisations à but non lucratif.

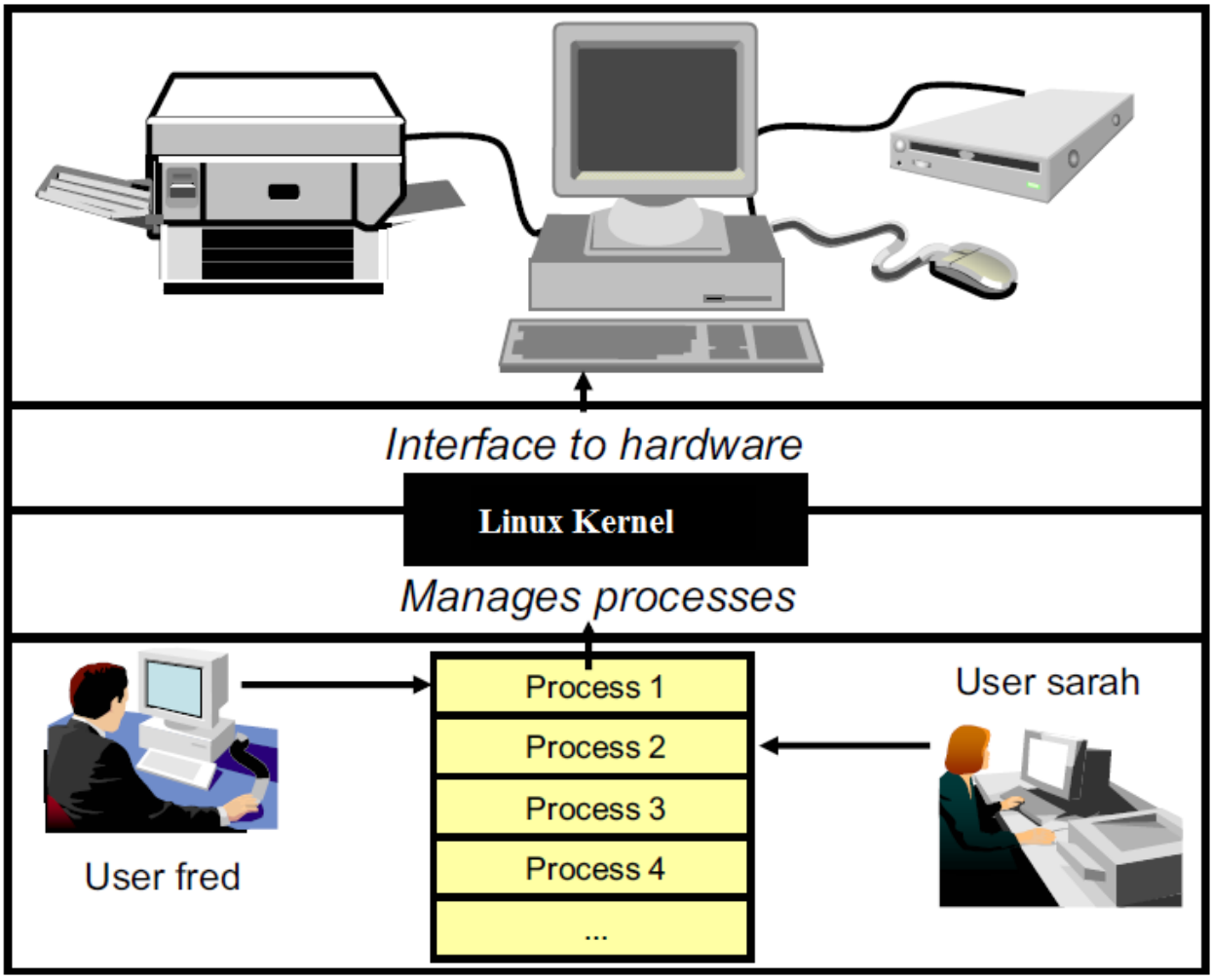
Exemples :

Linux		Unix	
Libre/Gratuit	**Payant**	**Gratuit**	**Payant**
CentOS	RedHat	FreeBSD	Oracle Solaris
Fedora	Suse	Oracle OpenSolaris	IBM AIX
Ubuntu	Slackware	…	HP UX
Debian	SCO Linux		IRIX
Opensuse	…		MacOS
Android			…
Mandriva			
…			**~Microsoft Windows**

N.B.: Les langages de programmation utilisés sur ces différent systèmes d'exploitation sont une combinaison de: C, C++, Assembleur.

II. Introduction à Linux :

4. Système d'exploitation linux :



5. Le Kernel Linux (noyau) :

Un ordinateur se compose de nombreux périphériques que les utilisateurs du système souhaitent utiliser. Par exemple, ils veulent imprimer des documents ou ils veulent jouer un jeu à partir d'un CD-ROM.

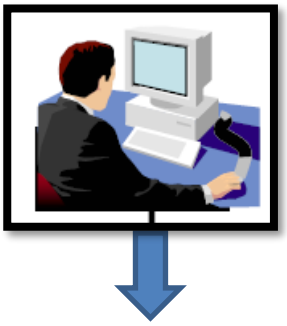
Pour contrôler ces périphériques et les partager entre plusieurs utilisateurs, un système d'exploitation doit être chargé au démarrage du système.

Dans le cas du système d'exploitation Linux, il y a un programme spécial qui s'interface directement avec le matériel ou les périphériques qui s'appelle le noyau (**Kernel**) Linux => **Le noyau contrôle l'accès aux périphériques.**

D'autre part, les utilisateurs peuvent démarrer de nombreux programmes, par exemple, un programme qui affiche un document ou supprime un fichier. Ces programmes qui s'exécutent sous format de processus Linux sont également contrôlés par le noyau Linux.

Pour le dire simplement : **Le noyau Linux (KERNEL) est le cœur de votre système d'exploitation**.

6. Exploiter le système Linux :



Login

Commande 1

Commande 2

Exploitation du SHELL

Who

Mail sarah

Communiquer avec les autres

Man mail

Help

Avoir de l'aide

Login :

Linux est un système d'exploitation multiutilisateur. Avant qu'un utilisateur commence à travailler sur Linux, il doit s'authentifier avec un nom d'utilisateur et un mot de passe.

SHELL :

Après l'authentification, Linux démarre un programme spécifique pour l'utilisateur qui s'appelle le **SHELL**.

Le SHELL est un interpréteur de commande qui attend une entrée (commande) et exécute l'entrée (commande) ou les programmes des utilisateurs.

Communication :

Plusieurs utilisateurs peuvent travailler en même temps sur un système Linux ou dans un réseau. L'une des tâches de base dans votre travail quotidien est de communiquer avec d'autres utilisateurs sur un système ou sur le réseau.

Informations complémentaires :

Linux propose une large gamme d'outils et de commandes. Il y a plusieurs façons d'obtenir l'aide sur ces commandes, par exemple, la commande '**man'**.

7. Structure des fichiers Linux :

L'une des tâches principales d'un système d'exploitation est de lire et écrire des données. Pour cela, Linux utilise une arborescence de fichiers hiérarchique qui se compose de répertoires, sous-répertoires et fichiers. Le répertoire de niveau supérieur est appelé **ROOT** (/) qui a plusieurs sous-répertoires.

Chacun des sous-répertoires peut contenir des fichiers ou sous-répertoires. Un répertoire est comme un dossier dans lequel vous mettez certains documents.

8. Types de systèmes de fichiers :

L'arborescence des fichiers est montée (MOUNT) lors du démarrage du système. Linux prend en charge les systèmes de fichiers de différents types, qui sont tous montés sur un arbre de fichier.

NB :

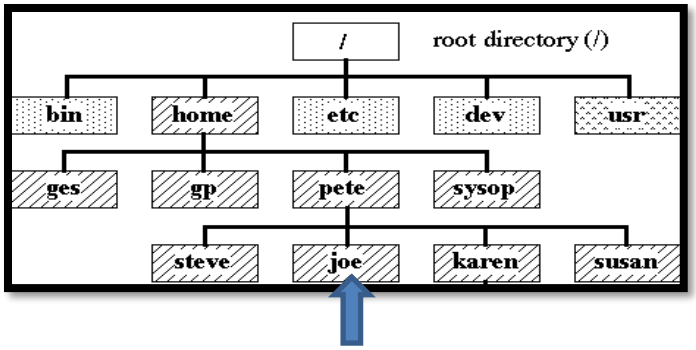
Il est strictement important de comprendre l'identification des composants d'un système d'exploitation Linux : **KERNEL, SHELL, FILE SYSTEM**.

9. Utilitaires Linux :

Deux composants que vous utiliserez sur Linux sont les fichiers et les répertoires. Pour travailler avec ces composants, Linux propose de nombreux services :

- La commande **find** pour rechercher des fichiers spécifiques
- La commande **grep** pour rechercher des modèles dans les fichiers (à l'intérieur)
- Commandes de comparaison de fichiers et de répertoires (**diff, cat**…)
- Commandes pour compresser et décompresser les fichiers afin d'économiser l'espace disque (**tar, gzip, zip**…)

Chercher un fichier



Comparer des fichiers

Compresser des fichiers

III. Exploitation de Linux :

1. Authentification :

Comme Linux est conçu comme un système multiutilisateur, un niveau de sécurité était mis en œuvre pour contrôler les accès. Chaque utilisateur du système a un nom d'utilisateur et un mot de passe associé (en option).

1.1. Nom d'utilisateur :

Lorsque le système est démarré et prêt pour un utilisateur, l'invite de connexion (généralement **Login :**) est présenté à l'écran. A ce stade, l'utilisateur doit fournir son nom d'utilisateur.

1.2. Mot de passe utilisateur :

Si le nom d'utilisateur nécessite un mot de passe, le système vous demandera le mot de passe de la même manière. Alors que l'utilisateur tape un mot de passe, **il n'apparaît pas sur l'écran**. Il est **fortement recommandé** d'utiliser des mots de passe pour tous les comptes d'utilisateurs.

Si le mot de passe utilisateur était mis en place par l'administrateur système, la première fois que l'utilisateur se connecte au système, l'utilisateur sera invité à modifier son mot de passe.

1.3. Connexion réussie :

Une fois connecté, l'utilisateur est présenté avec une invite (normalement commence par un **$**), **c'est le SHELL**.

1.4. Déconnexion :

Pour terminer la session, l'utilisateur peut saisir la commande « **exit** », « **logout** » ou appuyez sur la combinaison de touches <**CTRL+D**> (en maintenant la touche CTRL puis appuyer sur la touche d).

La déconnexion ne fonctionne que si vous êtes sur votre SHELL.

Lorsque l'utilisateur se déconnecte, après quelques secondes, une invite de connexion va apparaître sur l'écran.

1.5. Changer le mot de passe utilisateur :

Le mot de passe utilisateur est le principal mécanisme pour assurer la sécurité sur un système Linux.

Tous les mots de passe sont cryptés et ne peuvent pas être décodés par d'autres utilisateurs.

La commande « **passwd** » est utilisée pour changer le mot de passe utilisateur. (Un exemple d'une commande simple qui peut être entrée à l'invite du SHELL)

Le système démarre le processus **passwd** qui invite l'utilisateur à entrer son ancien mot de passe. Les caractères saisis ne sont pas affichés sur l'écran. Seulement si les deux correspondent => le nouveau mot de passe sera accepté. L'ancien mot de passe devient invalide par la suite.

Lorsque le processus **passwd** se termine l'utilisateur est de nouveau présenté par l'invite de commande.

2. Les commandes Linux :

Les commandes sous Linux ont le format suivant :

➢ **$ Commande Option(s) argument (s)**

2.1. Format d'une commande :

L'ordre et la séparation des éléments d'une commande sont importants. La commande ou le nom du processus doit être en premier. Les espaces sont utilisés par le SHELL comme séparateurs sur la ligne de commande et ne doit pas être placé dans le nom de la commande.

2.2. Les options d'une commande :

Les options doivent suivre le nom de la commande, séparés par un espace, et précédé d'un signe moins (-). Habituellement, plusieurs options peuvent être regroupées immédiatement après un seul signe moins ou séparés par des espaces et chacun est précédé par un signe moins. Les options sont généralement utilisées pour modifier le fonctionnement d'un processus.

2.3. Les arguments d'une commande :

Les arguments suivent les options et ils sont séparés par des espaces. L'ordre des arguments dépend de la commande.

2.4. Exemple :

$ ls	juste la commande
$ ls -l	commande et option
$ ls /tmp	commande et argument

2.5. Exceptions :

Quelques commandes sous Linux ne respectent pas le format précédent de la commande. Les commandes telles que « **tar** », « **date** » et « **ps** » acceptent des options qui ne commencent pas par le signe moins (-). Il s'agit d'assurer la rétrocompatibilité avec les anciennes implémentations Linux.

```
$ ps aux

$ date +%D
```

3. Les commandes date et cal :

Pour visualiser la date actuelle :

```
$ date
```

Pour voir le calendrier :

```
$ cal

$ cal 3 2003
```

4. Les commandes clear et echo :

Vider l'écran :

```
$ clear
```

Ecrire sur l'écran :

```
$ echo Déjeuner à 12h00
```

5. Les commandes who et finger :

Pour connaitre les utilisateurs actuels sur le système :

```
$ who
```

Afficher le nom d'utilisateur, le nom du poste de travail prolongée, heure de connexion et ID du processus de l'utilisateur courant :

```
$ who –u
```

Pour connaitre votre nom d'utilisateur :

```
$ who am i

$ whoami

$ who -m
```

Afficher des informations d'un utilisateur en ligne :

```
$ finger user3
```

6. Envoyer un mail :

La commande « **mail** » est une commande interactive utilisée pour envoyer et recevoir des messages électroniques.

Pour envoyer un mail :

```
$ mail user2
```

Le système vous présente un message « **Subject** » dont vous devez fournir l'objet du message, après cela vous pouvez saisir votre message à envoyer.

Pour finir le message, cliquez sur « **Entrée** » puis « **CTRL+D** ».

Le système vous présente avec un nouveau champ « Cc » afin d'ajouter d'autres utilisateurs comme destinataire (en option).

La commande mail vous permet aussi d'envoyer des messages sur le réseau :

```
$ mail user3@hôte2
```

7. Recevoir un mail :

L'utilisateur est informé quand il reçoit des nouveaux messages [You have new mail]. Ce message n'est pas affiché automatiquement dès que l'e-mail arrive. Le SHELL fait une vérification sur toutes les boîtes aux lettres par défaut une fois toutes les 600 secondes. S'il détecte un nouveau message, il affiche le message.

Pour lire le message :

```
$ mail
```

Il donnera la liste des informations d'en-tête et une description de ligne pour chaque élément non lu suivie de l'invite « **?** », ceci est différent de l'invite du SHELL.

8. Gestion de messagerie électronique :

À cette invite, l'utilisateur peut entrer l'une des commandes du sous-système de messagerie.

Certaines de ces commandes qui peuvent être utilisées à l'invite sont les suivants :

- **d** suppression
- **m** transférer le message
- **R** envoyez une réponse à l'expéditeur de messages dans la file d'attente
- **q** quitter la boite de messagerie et garder les messages en la file d'attente
- **s** ajouter des messages dans un fichier
- **t** afficher un message
- **x** quitter la boite de messagerie et annuler toutes les modifications de la file d'attente

Certaines sous-commandes telles que "**d**" et "**t**" acceptent des numéros de messages comme arguments.

9. Les commandes write et wall :

La commande « **write** » peut être utilisé pour envoyer des messages à **un** utilisateur de ce système ainsi que les utilisateurs des autres systèmes connectés au réseau.

La commande « **wall** » envoie des messages à tous les utilisateurs.

Les deux commandes « **write** » et « **wall** » vont uniquement envoyer des messages aux utilisateurs qui sont en ligne.

Afin d'envoyer un message aux autres systèmes sur le réseau :

```
$ write user3@hôte2
```

10. La commande man :

La commande man va chercher dans le manuel système pour plus d'informations sur les commandes. Ces informations seront présentées page par page à l'utilisateur.

Les informations qui seront affichées :

Purpose Le titre et une description courte de la commande

Syntax Une liste de toutes les options et les arguments valides

Description Plusieurs pages contenant des informations sur la fonction et l'utilisation de la commande avec des exemples

Flags Les options disponibles

Examples Exemples d'utilisation de la commande

Files Tous les fichiers système associés à la commande

Related Info. Les noms des commandes associées

IV. Fichiers et Répertoires :

1. Fichiers :

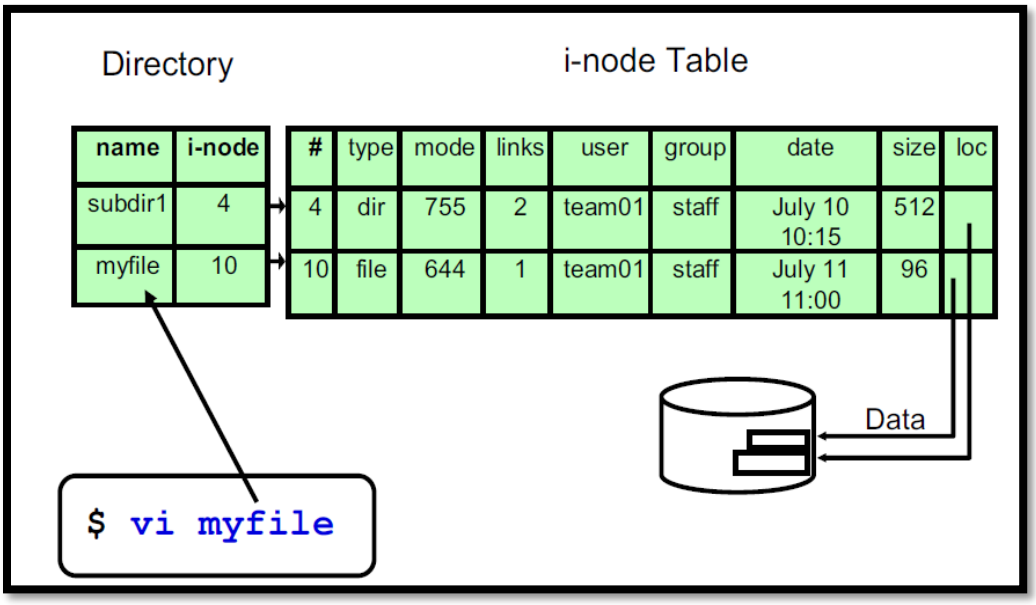
Les fichiers sont des collections de données ou de caractères.

Types de fichiers :

- Ordinaire : Texte
- Répertoire : une table de contenu qui stocke une liste de fichiers à l'intérieur de ce répertoire.
- Fichiers spéciaux : Représente le matériel ou bien des équipements logiques.
    - Exemple : CD-ROM est accessible via le fichier **/dev/cd0**.

2. Répertoires :

Les répertoires vous permettent de regrouper des fichiers et sous-répertoires. Un répertoire est un type unique de fichier qui contient suffisamment d'informations pour relier un nom de fichier à l'i-node qui décrit le fichier. En conséquence, les répertoires occupent généralement moins d'espace que les fichiers ordinaires.



2.1. Contenu des répertoires :

Chaque entrée de répertoire contient un nom de fichier ou un sous-répertoire et son numéro i-node associé.

2.2. L'accès des utilisateurs aux fichiers :

Lorsqu'un utilisateur exécute une commande pour accéder à un fichier, il utilisera le nom du fichier. Le système compare ensuite le nom du fichier correspondant avec le numéro d'i-node.

Une fois que le numéro d'i-node est connu, le système accédera à une table d'i-node, qui contient des informations sur les caractéristiques du fichier.

2.3. i-node:

Des exemples de ce qui est stocké dans la table des i-nodes inclus l'ID du propriétaire du fichier, le type de fichier, la date du dernier accès au fichier et la dernière modification, la taille du fichier et l'emplacement du fichier. Une fois que le système connaît l'emplacement du fichier, les données réelles peuvent être localisées.

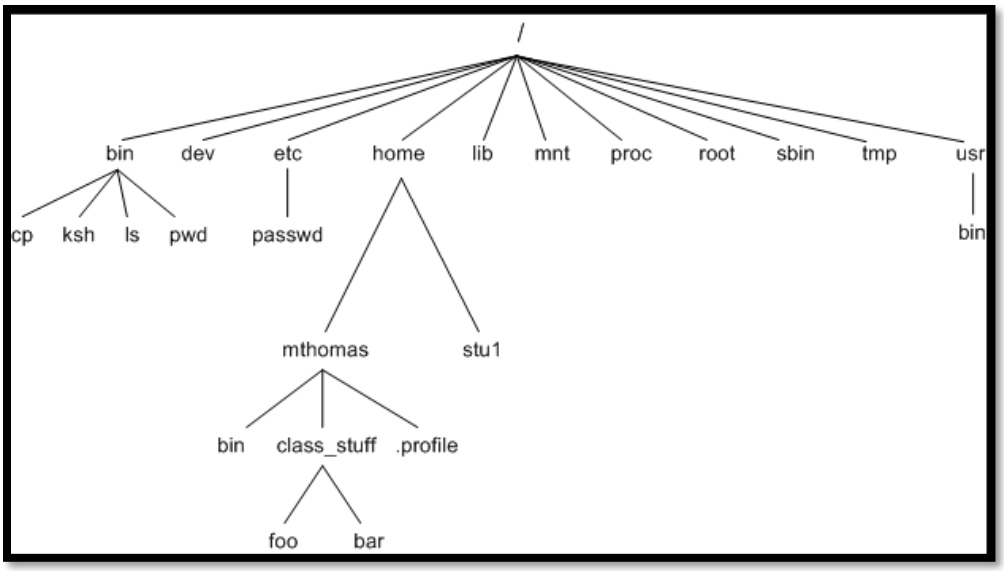
3. Système de fichier (File System) :

Le terme système de fichier (File System) désigne à la fois l'emplacement physique de données ainsi que l'organisation logique des répertoires pour permettre un rangement rapide et facile ainsi que la récupération de ces données.

Un système de fichier physique est une allocation de stockage sur un dispositif tel que : un disque dur ou un lecteur de CD-ROM. Le concept est très similaire à des partitions sur certains systèmes d'exploitation.

L'espace est attribué sur un périphérique et il est associé à un identifiant que l'utilisateur peut utiliser pour accéder aux fichiers. Dans Linux, cet identifiant est un répertoire dans la structure de l'ensemble du système de fichiers logique.

Si un système de fichiers physique se remplit, aucun fichier supplémentaire ne peut être créé au sein du système.



Exemples :

Quelques répertoires classiques qui peuvent être trouvés sur le répertoire **ROOT** (/) :

- **/sbin** - Utilitaires pour le démarrage du système

- **/dev** - fichiers spéciaux qui représentent les périphériques
- **/etc** - les fichiers de configuration du système
- **/usr**- contient les programmes, les bibliothèques et les fichiers de données des applications Linux.
- **/home** -contient les répertoires de connexion et les fichiers d'utilisateurs.
- **/var** - contient des fichiers qui changent dynamiquement.
- **/tmp** -contient les fichiers temporaires créés par l'exécution des programmes.
- **/opt**– contient les applications tierces. un exemple des fichiers de ce répertoire sont les commandes de base de Linux, tels que tar, gzip, gunzip, bzip2…
- **/proc** –Linux utilise ce répertoire pour mapper les processus et les structures de données du noyau aux fichiers correspondants.

4. Les chemins d'accès aux fichiers :

Le chemin d'accès à un fichier est écrit comme une chaîne de noms séparés par des slaches (/). Le dernier nom à droite est le nom de fichier auquel vous souhaitez accéder. Les autres noms sont les répertoires et sous-répertoires qui décrivent comment se rendre à l'emplacement du fichier.
Le chemin d'accès à un fichier est toujours considéré comme relatif à moins que cela commence par une slache.

Exemples :

```
vi   /home/team01/doc/mon_report

/usr/bin/ls   –l   /home/team01
```

5. Utilisation de la commande pwd :

La commande « **pwd »** renvoie toujours le chemin complet de votre répertoire de travail actuel. Cette commande est utilisée souvent par les utilisateurs Linux.

Exemple :

```
$ pwd

/home/team01
```

6. Utilisation de la commande ls :

La commande « **ls** » permet de lister le contenu d'un répertoire et dispose de nombreuses options utiles. Si aucun nom de fichier ou répertoire n'est spécifié comme argument de la commande « **ls** », le répertoire courant sera utilisé.

Par défaut, la commande « **ls** » affiche les informations dans l'ordre alphabétique. Lorsque la commande « **ls** » est exécutée, elle n'affiche pas les noms de fichiers commençant par un point « **.** » (Il faut utiliser l'option **-a**). Ces fichiers sont généralement appelés fichiers cachés. Pour lister tous les sous-répertoires et leur contenu, l'option-**R** peut être utilisé.

Exemple :

```
$ ls

c doc manuals test1

$ ls –a

. .. .profile c doc manuals test1

$ ls –R

c doc manuals test1

./c:

./doc:

Mon_report trio_ltr walrus
```

Affichage détaillé de la commande ls :

Les champs de la commande « **ls -l** » sont les suivantes:

Permissions	Liens	Propriétaire	Groupe	Taille	Dernière modification	Nom du fichier
drwxrwxr-x	2	team01	staff	1024	Aug 12 10:16	c

7. Utilisation de la commande cd :

La commande « **cd** » permet de changer le répertoire de travail courant.

L'utilisation de la commande « **cd** » sans aucun paramètre ou attribut vous déplacerez vers votre répertoire personnel.

8. Création et suppression d'un répertoire :

Pour créer un nouveau répertoire on utilise la commande Linux « **mkdir** ».

Exemple :

```
mkdir /home/team01/tesmkdir /home/team01/test
```

NB : Utilisez la commande avec l'option « -p » pour créer plusieurs répertoires simultanément.

Pour supprimer un répertoire, on utilise la commande Linux « **rmdir** ». (Le répertoire doit être vide)

Exemple :

```
rmdir /home/team01/tesmkdir /home/team01/test
```

NB : Utilisez la commande avec l'option « **-p** » pour supprimer plusieurs répertoires simultanément. (Les répertoires doivent être vides)

9. Création d'un fichier :

Pour créer un nouveau fichier on utilise la commande Linux « **touch** ».

Exemple :

```
touch   /home/team01/tesmkdir   /home/team01/test
```

V. Utilisation des fichiers :

1. Copie :

La commande Linux « **cp** » est utilisée pour faire une copie d'un ou plusieurs fichiers source vers un fichier de destination. (Fichier source peut être un dossier, option « **-R** »).

NB : si le fichier de destination existe déjà, Linux va écraser l'ancien fichier pour éviter cela utiliser l'option « -i » avec la commande.

Exemple :

cp doc/programa test1 c (afin de copier plusieurs fichiers sources, la destination doit être un répertoire)

cp –R a /test (on utilise l'option « -R » afin de copier un répertoire)

2. Déplacer et renommer :

La commande Linux « **mv** » est utilisé pour déplacer les fichiers et répertoires d'un répertoire à un autre ou bien pour renommer un fichier ou un répertoire. Si vous déplacez un fichier ou un répertoire vers un nouveau répertoire, il conserve le nom de fichier de base. Lorsque vous déplacez un fichier, tous les liens vers d'autres fichiers restent intacts, sauf si vous déplacez vers un système de fichiers différent. Lorsque vous déplacez un répertoire dans un répertoire existant, le répertoire et son contenu sont ajoutés dans le répertoire existant.

NB : si le fichier de destination existe déjà, Linux va écraser l'ancien fichier pour éviter cela utiliser l'option « -i » avec la commande.

Exemple :

mv t.letter ../doc/letter (déplace le fichier « t.letter » vers « ../doc/letter)

NB :

- **« .. » signifie le répertoire parent**
- **« . » signifie le répertoire courant**

3. Contenu d'un fichier :

Pour afficher le contenu d'un fichier, on utilise la commande Linux « **cat** ».

La commande « **cat** » peut être exécuté avec l'option « **-n** » pour afficher les numéros de lignes.

NB : La commande « **cat** » n'effectue pas la pagination du contenu d'un fichier. Si le contenu du fichier dépasse la taille de l'écran, seulement la fin du fichier qui sera affichée sur l'écran.

Afin d'afficher le contenu d'un fichier dans un format paginer, vous pouvez utiliser la commande :

```
more
```

4. La commande « wc » :

La commande Linux « **wc** » permet de compter le nombre de lignes, de mots et caractères dans un fichier nommé.

Option :

- « **-l** » compter le nombre de lignes
- « **-w** » compter le nombre de mots
- « **-c** » compter le nombre de caractères

5. Liaison de fichiers :

La commande Linux « **ln** » permet d'avoir un fichier avec deux ou plusieurs noms différents dans la même structure de répertoire. Ces références multiples sont également connues comme des liens durs (hard). Aucune copie du fichier n'aura lieu, chaque référence dans le répertoire pointe vers le même **i-node**.

Lorsque vous utilisez la commande « **ln** », la source est le fichier existant alors que la cible est la nouvelle référence de fichier (ou lien) qui doit être créé.

NB : Ce type de lien peut uniquement être créé sur le même système de fichier.

Exemple :

Pour créer un lien :

```
ln   source_file   target_file
```

Pour afficher les liens d'un fichier :

```
ls -li
```

Lien symbolique :

La commande « **ln** » permet également la création de liens indirects aux fichiers. Ceux-ci s'appellent des liens symboliques, et sont créés en utilisant l'option « -n ».

Exemple :

Pour créer un lien :

```
ln   –s   source_file   target_file
```

6. Supprimer un fichier :

La commande « **rm** » permet de supprimer un fichier spécifique ou les fichiers d'un répertoire. Par défaut, il n'y a pas de confirmation avant la suppression. Pour inclure une confirmation, on utilise l'option « **-i** » (interactive).

On utilise l'option « **-R** » pour supprimer un répertoire et son contenu.

Exemple :

```
rm mon_fichier

rm –i mon_fichier

rm –R mon_repertoire
```

VI. Les permissions des fichiers :

1. Liste de fichier :

La commande « **ls** » avec l'option « **-l** » peut être utilisée pour obtenir plus d'informations sur les fichiers dans un répertoire.

NB : Si on utilise l'option « **-ld** » cela vous permettrez d'afficher uniquement les informations concernant les répertoires.

Exemple :

- **ls –l**

1	2	3	4	5	6	7
drwxrwxr-x	2	team01	staff	512	Feb 1809:55	doc
-rwxrwxr-x	1	team01	staff	320	Feb 2207:30	suba

Les différents champs affichés par la commande « **ls** » sont :

1) Les bits de permissions d'un fichier ou répertoire
2) Compteur de liens
3) Le nom de l'utilisateur propriétaire du fichier ou répertoire
4) Le nom du groupe propriétaire
5) Le nombre de caractères
6) La date et l'heure de la dernière modification
7) Le nom du fichier ou répertoire

2. Les permissions :

- Chaque fichier et répertoire sur le système possède des permissions
- Trois (3) catégories d'autorisations : le propriétaire (owner), le groupe (group) et les autres (others)
- Trois bits peuvent être définis pour chaque catégorie : lecture, écriture, exécution (rwx)

Fichier :

R	consulter le contenu d'un fichier
W	modifier ou supprimer le contenu d'un fichier
X	exécuter le fichier (**r** est également nécessaire)

Répertoire :

r	consulter le contenu d'un répertoire (fichiers et sous répertoires)
w	modifier ou supprimer le contenu d'un répertoire (fichiers et sous répertoires)
x	Se déplacer sur le répertoire et accéder aux fichiers

3. Modifications des permissions : (Notation symbolique)

Avec la notation symbolique, vous spécifiez les changements relatifs aux permissions existantes sur un fichier ou un répertoire en ajoutant ou en supprimant les autorisations avec la commande « **chmod** ».

NB : Vous pouvez vérifier les permissions actuelles à l'aide de la commande « ls -l ».

chmod mode nom_de_fichier

u => utilisateur propriétaire	• **+ : ajouter permission**
g => groupe	• **- : supprimer permission**
o => autres	
a => tout le monde	

Exemple :

```
$ ls -l newfile
-rw-r--r-- 1   team01   staff    58 Apr 21  16:06      newfile

$ chmod go+w newfile
$ ls -l newfile
-rw-rw-rw- 1   team01   staff    58 Apr 21  16:06      newfile

$ chmod a+x newfile
$ ls -l newfile
-rwxrwxrwx 1   team01   staff    58 Apr 21  16:06      newfile

$ chmod o-rwx newfile
$ ls -l newfile
-rwxrwx--- 1   team01   staff    58 Apr 21  16:06      newfile
```

4. Modifications des permissions : (Notation octal)

Chaque permission dans un groupe de neuf (9) est représenté par **(1)** quand une permission existe et par **(0)** dans le cas échéant.

Alors « **rw- r-- r--** » se traduit « **110 100 100** » en binaire et « **644** » en notation octal.

	Utilisateur	**Groupe**	**Autres**
Symbolique	rwx	rw-	r--
Binaire	111	110	100
Octal	4+2+1 7	4+2+0 6	4+0+0 4

Exemple :

```
$ ls -l newfile
-rw-r--r--  1  team01  staff  58 Apr 21 16:06  newfile

$ chmod 664 newfile
$ ls -l newfile
-rw-rw-r--  1  team01  staff  65 Apr 22 17:06  newfile
```

5. Les permissions par défaut :

Les permissions par défaut pour les fichiers et les répertoires sont :

- **Fichier : 644**
- **Répertoire : 755**

Ces permissions par défaut sont spécifiées par un paramètre système appelé « **umask** ».

Les véritables permissions par défaut pour un fichier et un répertoire sont 666 et 777 respectivement. La valeur de « **umask** » est ensuite soustraite de ces valeurs. Le « **umask** » par défaut pour un système Linux est « **022** », ce qui vous laisse avec les valeurs de « 644 » pour un fichier et 755 pour un répertoire.

6. Les permissions requises :

Command	Répertoire Source	Fichier source	Répertoire Destination
cd	x	N/A	N/A
ls	r	N/A	N/A
ls -l	r, x	N/A	N/A
mkdir	x w (Parent)	N/A	N/A
rmdir	x w (Parent)	N/A	N/A
cat, pg, more	x	r	N/A
mv	x, w	N/A	x, w
cp	x	r	x, w
touch	x, w*	N/A	N/A
rm	x,w	N/A	N/A

VII. L'éditeur de texte vi :

Il est important de connaitre l'éditeur de texte « **vi** » car :

- ✓ Le seul éditeur accessible sur la plupart des OS Linux et UNIX
- ✓ Editeur par défaut pour la majorité des programmes
- ✓ Interface en deux modes Commande et Texte

L'éditeur vi est accessible en deux modes :

- ➢ Commande : déplacé, copié, collé, suppression des lignes…
- ➢ Texte : insérer du texte.

1. Démarrage de « vi » :

Pour démarrer l'éditeur « **vi** » :

vi nom_du_fichier

NB :

- ➢ Si le fichier « **nom_du_fichier** » n'existe pas, l'éditeur va le créer.
- ➢ Si le fichier « **nom_du_fichier** » existe, l'éditeur démarra le fichier en écriture.

2. Ajouter du texte :

Pour ajouter du texte sur le fichier, les commandes suivantes peuvent être utilisées :

a	=> Ajouter le texte **APRES** le curseur
A	=> Ajouter le texte à **LA FIN** de la ligne
i	=> Insérer le texte **SUR** le curseur
I	=> Insérer le texte **SUR** le début de la ligne
o	=> Ouvrir une nouvelle ligne **SOUS** la ligne courante
O	=> Ouvrir une ligne **EN HAUT** de la ligne courante

Pour quitter le mode texte vers le mode commande, taper : **<Esc>**

3. Arrêter l'éditeur « vi » :

Pour entrer dans le mode commande ou bien pour s'assurer que vous êtes en mode commande, appuyez sur **<Esc>** avant d'effectuer les commandes d'arrêt de l'éditeur.

Les commandes suivantes vont arrêter l'éditeur :

:q	=> quitter sans enregistrer

:w	=> enregistrer
:wq	=> enregistrer et quitter
:x	=> enregistrer et quitter
Shift+zz	=> enregistrer et quitter

NB : La commande « : » place le curseur en bas de l'écran.

4. Mouvement du curseur :

Dans une ligne :

- ✓ **h** => un caractère à gauche
- ✓ **l** => un caractère à droite
- ✓ **0** => au début de la ligne
- ✓ **$** => à la fin de la ligne

Dans l'écran :

- ✓ **k** => une ligne en haut
- ✓ **j** => une ligne en bas
- ✓ **H** => la première ligne sur l'écran
- ✓ **M** => la ligne au milieu de l'écran
- ✓ **L** => la dernière ligne sur l'écran
- ✓ **G** => la dernière ligne
- ✓ **45G** => la ligne 45
- ✓ **Ctrl+f** => défilé en avant
- ✓ **Ctrl+b** => défilé en arrière

5. Supprimer du texte :

- ➢ **x** => supprimer un caractère
- ➢ **dd** => supprimer une ligne
- ➢ **u** => annuler le dernier changement

6. Rechercher du texte :

En mode commande :

- ➢ **/test** => recherché en avant l'occurrence du mot 'test'
- ➢ **?test** => recherché en arrière l'occurrence du mot 'test'

NB : taper « n » pour continuer la recherche sur la même direction.

7. Remplacer du texte :

Pour remplacer un mot ou bien une chaine de mot par une autre il faut utiliser le modèle suivant :

:g/ the /s// the one and the only /g

Cela va remplacer le mot « **the** » avec la chaine « **the one and the only** ».

8. Couper, Copier et coller :

- **dd** => couper une ligne (10dd couper les 10 premiers lignes à partir du curseur)
- **yy** => copier une ligne (10yy copier les 10 premiers lignes à partir du curseur)
- **p** => coller la ligne

VIII. Les bases du SHELL :

Le **SHELL** est l'interface principale entre l'utilisateur et le noyau du système d'exploitation. Le **SHELL** standard dans la plupart des systèmes UNIX est « **ksh** » (Korn SHELL) et pour Linux « **bash** » (Bourne Again SHELL).

Le **SHELL** interprète les commandes de l'utilisateur pour lancer des applications et exploiter les utilitaires systèmes pour la gestion des données.

Le **SHELL** peut être utilisé comme un langage de programmation globale en combinant des séquences de commandes avec des variables.

1. Substitution des noms de fichiers :

La substitution sous Linux est assurée par les méta-caractères (**?, *, /, -, etc**).

1.1. Les caractères ? Et * :

? : est étendu par le **shell** pour correspondre à un caractère unique dans un nom de fichier. L'exception est que le caractère « **?** » ne correspondra pas à un point « **.** » comme premier caractère d'un nom de fichier (par exemple, un fichier caché).

Exemple :

```
ls  ne?

net  new
```

***** : est étendu par le **shell** pour correspondre à n'importe quel nombre de caractères dans un nom de fichier. Le caractère « * » sera étendu pour signifier tous les fichiers dans le répertoire courant, sauf ceux qui commencent par un point.

Exemple :

```
cp    n*    /tmp

ne  net  new  nest
```

NB : Méfiez-vous de la commande « **rm *** » qui pourrait causer la suppression de tous les fichiers du répertoire courant.

1.2. Les listes d'inclusions :

La position tenue par les crochets **« [et]** » et leur contenu sera adapté à un seul caractère dans le nom du fichier. Ce caractère sera soit un membre d'une liste ou d'une plage, ou tout caractère qui n'est pas un membre de cette liste si la plage contient le caractère **« ! »** au début de la liste d'inclusion.

[az] : représente les lettres « a » ou « z »

[a-z]	**:** représente les lettres de « a » à « z »
[!az]	**:** représente tous les caractères sauf « a » ou « z »

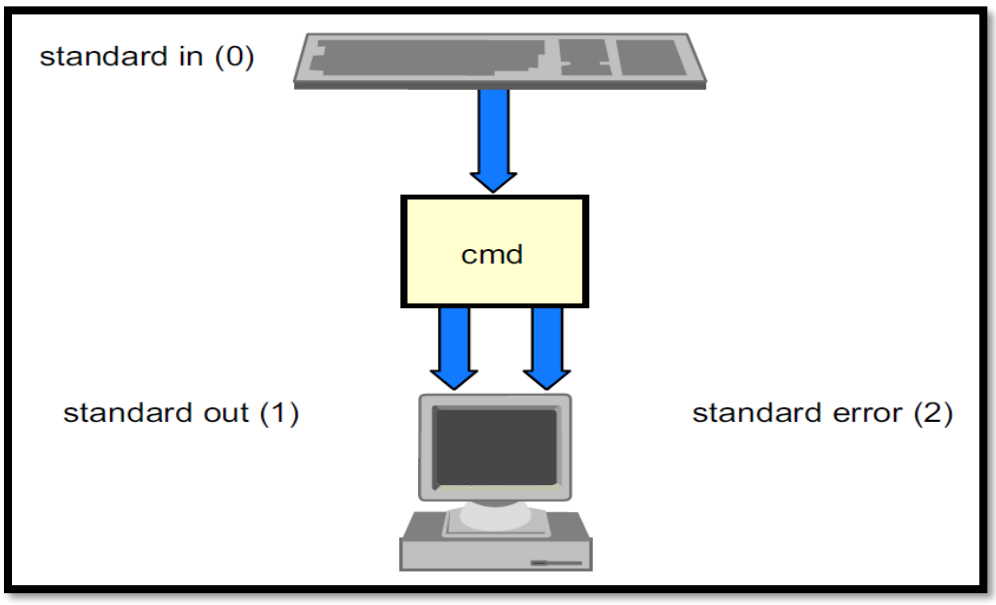
Exemple :

ls	**ne[stw]**
net new	
rm	**[fghjdn]e[tw]**
few net new	
ls	***[1-5]**
test1 test1.2 test1.3	
cat	**?[!y]*[2-5]**
test1.2 test1.3	

1.3. Redirection de commande :

Trois fichiers sont automatiquement ouverts pour chaque processus sur le système. Ces fichiers sont appelés :

- l'entrée standard
- sortie standard
- l'erreur standard



Lorsqu'une application utilise un fichier, il ouvre le fichier en utilisant le chemin d'accès au fichier, mais une fois que le fichier est ouvert l'application utilise un identifiant numérique, appelé un descripteur de fichier pour identifier le fichier pour la lecture ou pour l'écriture. Les chiffres indiqués sur la feuille sont les descripteurs de fichier standard pour : **STDIN (0) « < », STDOUT (1) « > », et STDERR (2) « 2> »**.

L'entrée standard, parfois abrégée « **stdin** », c'est où une application obtient son entrée. C'est généralement le clavier ou un fichier.

La sortie standard « **stdout** » et l'erreur standard « **stderr** » sont où une application envoie sa sortie (ou erreur). C'est normalement l'écran ou un autre fichier.

Input :

```
cat datafile.txt | grep abc
```

Lire le fichier « **datafile.txt** » et rediriger la sortie vers la deuxième commande « **grep** » afin de trouver l'occurrence de la chaine « **abc** ».

Output :

```
ls > ls.output
```

Rediriger la sortie de la commande « **ls** » vers le fichier « **ls.output** ».

NB : Si le fichier « **ls.output** » existe déjà, le contenu du fichier sera écrasé (pour ajouter le contenu au fichier sans supprimer utilisez la redirection « >> »).

Erreur :

Sans redirection d'erreur :

```
cat filea fileb

This is output from filea.

cat: cannot open fileb
```

Avec redirection de l'erreur :

```
cat filea fileb 2> errfile
```

2. Pipe :

Deux ou plusieurs commandes peuvent être séparées par un « **pipe** » sur une seule ligne de commande. La sortie standard de chaque commande à gauche du « **pipe** » devient l'entrée standard de la commande à droite du « **pipe** »

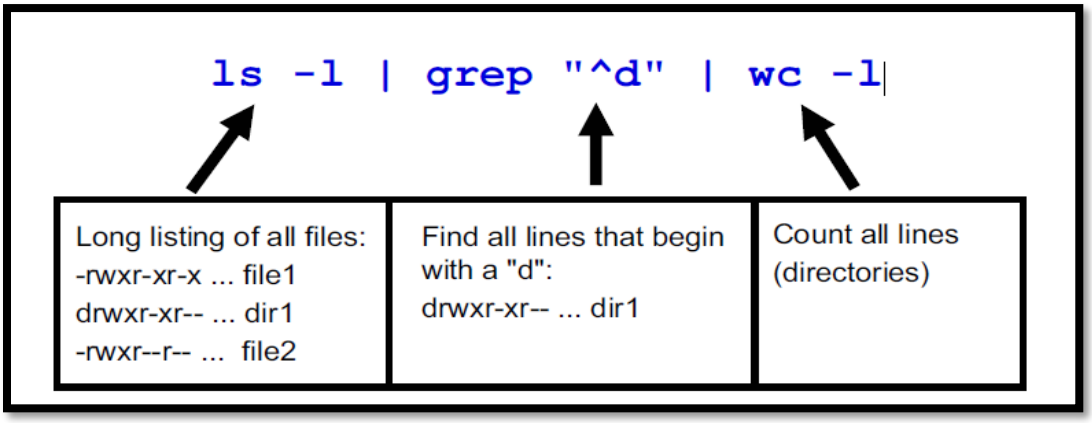
Exemple :

```
who | wc -l
4
```

3. Filtre :

Un filtre est une commande qui lit l'entrée standard, transforme l'entrée d'une certaine façon, et écrit sur la sortie standard.

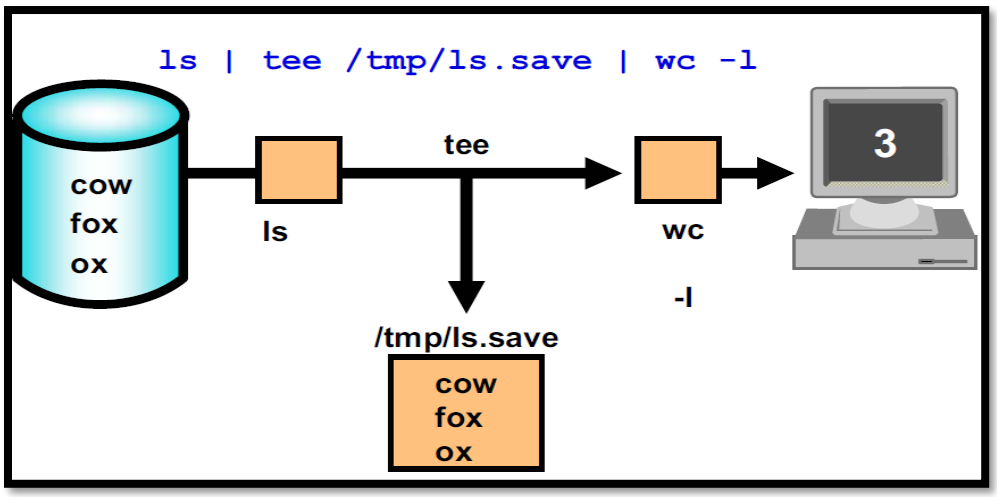
Exemple :



4. La commande « tee » :

La commande « **tee** » est un filtre qui peut être utilisé pour effectuer une capture instantanée de l'information passant par un « **pipe** ». « **tee** » met une copie des données dans un fichier, ainsi qu'il fait passé la sortie standard de la première commande pour être utilisé par la commande suivante.

Exemple :



5. Regroupement de commandes :

Plusieurs commandes peuvent être saisies sur la même ligne, séparée par un point-virgule "**;**":

Exemple :

```
ls  -R  >  outfile  ;  exit
```

6. Continuation de ligne :

Le **backslash** « \ » suivi est utilisée pour poursuivre une commande sur une ligne distincte.

Exemple :

Après le **backslash** il faut taper « **Entrée** » :

```
cat /home/mydir/mysubdir/mydata \

> /home/yourdir/yoursubdir/yourdata
```

IX. Les variables SHELL :

Les variables représentent des données dont la valeur peut changer.

Tous les variables **shell** sont sensibles. Par exemple, **HOME** et **home** sont deux différents variables.

Par convention, les noms en majuscules sont utilisés pour les variables standards définis par le **système** et les noms en minuscules sont utilisés pour les variables définies par **l'utilisateur**.

1. Lister les paramètres des variables :

La commande « **set** » affiche les noms et les valeurs de toutes les variables **shell**. La commande « **set** » est une commande intégrée du **shell**, et donne donc une sortie différente en fonction du **shell** en cours d'exécution, par exemple **Bourne** ou **Korn shell**.

Exemple :

```
set

HOME=/root

HOSTNAME=localhost.localdomain

HOSTTYPE=x86_64
```

2. Configuration d'une variable :

Les variables peuvent contenir n'importe quel type de données, comme des nombres entiers, des mots simples, des chaînes de caractère ou des nombres complexes.

1. Pour assigner une valeur à une variable :

```
name=value
```

2. Pour se référer aux variables on préfixe son nom par « **$** » :

```
xy= "hello world"

echo $xy

hello world
```

3. Pour supprimer une variable :

```
unset xy
```

Exemple :

xy=day

echo $xy

day

echo Tomorrow is Tues$xy

Tomorrow is Tues**day**

echo There will be a $xylong meeting

There will be a meeting

echo There will be a ${xy}long meeting

There will be a **day**long meeting

NB :

Il ne faut pas mettre un espace avant le « **$** » de la variable.

4. Substitution des commandes :

- **date**
- Wed 11 Jul 11:38:39 2003
- **now=$(date)**
- **echo $now**
- Wed 11 Jul 11:38:39 2003
- **HOST=$(hostname)**
- **echo $HOST**
- sys1
- **echo "Today is $now and `who | wc -l` users **
  > are logged in"
- Today is Wed 11 Jul 11:45:27 2003 and 4 users are logged in

Une variable peut être fixée à la sortie de certaines commandes ou groupe de commandes en utilisant les accents graves. Ils ne doivent pas être confondus avec des guillemets simples. Dans les exemples, la sortie de la date est mémorisée dans une variable.

5. Les méta-caractères :

5.1. Apostrophe :

Ignorer toutes les méta-caractères entre les apostrophes.

echo '$HOME'

$HOME

5.2. Guillemet :

Ignorer toutes les méta-caractères entre les guillemets sauf **$, ` et \.**

echo “$HOME”

/usr/local

5.3. Backslash :

Ignorer la signification du méta-caractère suivant :

echo \$HOME

$HOME

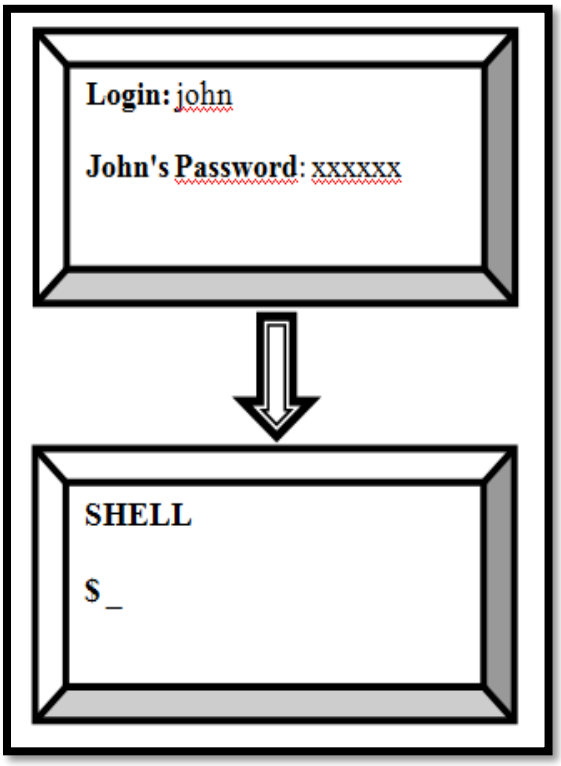
X. Les processus :

Un programme ou une commande en exécution sur le système est appelé un processus. Sous Linux, on peut exécuter plusieurs processus en même temps, ainsi que de nombreuses occurrences d'un même programme (tel que vi) simultanément dans le système.

Pour lister les processus lancer par un utilisateur :

```
ps    -u    user01
```

1. Le processus d'environnement :



Environnement

Programme **/bin/sh**

UID **john**

GID **staff**

Fichiers **/dev/tty1**

Lorsque vous vous connectez à un système, Linux démarre un nouveau processus (exemple avec PID = 202) et charge le programme **SHELL** « **/bin/sh »** dans ce processus. Ce **SHELL** est appelé le **SHELL** de connexion.

Le PID (Process ID) est attribué au hasard par le **KERNEL**.

2. Les relations processus :

Les processus existent dans une relation d'hiérarchies « **parent/fils** ». Un processus qui démarre un programme ou une commande est le processus **parent** et un processus **fils** est le produit du processus parent. Un processus parent peut avoir plusieurs processus fils, mais un processus fils ne peut avoir qu'un parent.

Ce processus se manifeste lorsque l'utilisateur commence l'exécution des commandes après qu'ils soient connectés au système. Le **shell** attend les instructions et les exécutent. Les instructions impliquent généralement le démarrage d'un processus, comme un éditeur.

Dans cette situation, le **shell** est le processus parent et l'éditeur devient le fils.

3. Les variables et processus :

Les variables d'environnement sont locales au **shell** dont les processus fils n'héritent pas automatiquement les variables parents.

Pour passer des variables dans un **sous-shell,** la commande d'exportation doit être exécutée.

Activité :

Cette activité présente la commande d'exportation.

1. Connectez-vous au système :
2. Notez l'ID du processus de votre shell courant :

echo $$

3. Définissez deux variables shell **vartest1** et **vartest2** de la façon suivante :

vartest1 = "lune"

vartest2 = "mars"

4. Exécutez la commande d'exportation seulement pour **vartest2**.

export vartest2

5. Affichez les valeurs des variables vartest1 et vartest2 :

echo $vartest1

echo $vartest2

6. démarrez un nouveau shell :

/bin/sh

7. Affichez l'ID du processus du nouveau shell :

echo $$

8. Affichez les valeurs des variables vartest1 et vartest2 :

echo $vartest1

echo $vartest2

4. Script SHELL :

Un script **shell** est un fichier texte simple qui contient des commandes Linux.

Quand un script **shell** est exécuté, le **shell** lit le fichier ligne par ligne et traite les commandes dans l'ordre logique.

Invoquer un script SHELL :

1. Ouvrez un fichier texte par l'éditeur « **vi** » (exemple) et insérez la portion du texte suivante :

```
echo "Hello, John. Today is: $(date)"

pwd

ls
```

2. Afin d'exécuter ce script il faut exécuter la commande suivante :

```
sh script
```

Le **shell** lit le script ligne par ligne et exécute les commandes ligne par ligne. L'invocation du script fait appel au **shell** pour permettre l'exécution.

L'exécution d'un script peut être faite en faisant appel au script avec son nom :

```
/root/script
```

Le **shell** vous retourne un message d'erreur, cela revient aux autorisations affectées au script :

```
ls  –l  script

-rw-r--r--.  1  root  root  46 Dec 11 10:49  script
```

Vous pouvez remarquer que le fichier n'a pas de droit d'exécution, pour cela on modifie les autorisations du fichier en ajoutant le droit d'exécution :

```
chmod +x script

ls  –l  script

-rwxr-xr-x.  1  root  root  46 Dec 11 10:49  script
```

Exécutez le script actuellement avec son nom :

```
/root/script
```

Vous pouvez remarquer maintenant que le script s'exécute sans problème.

NB :

On utilise le chemin absolu pour exécuter le script **(/root/script)**.

Pour qu'on puisse l'exécuter de n'importe quel endroit il faut ajouter le répertoire contenant le fichier du script au chemin du variable d'environnement **$PATH**.

Modifiez le fichier « **.bash_profile** » (fichier caché) qui se trouve sur votre répertoire HOME :

```
cd $HOME

ls –la
```

NB : on utilise l'option « -a » pour afficher les fichiers cachés.

```
vi  .bash_profile
```

Ajoutez le chemin vers votre script de test « **script** » :

```
:/home
```

NB : on utilise « **:** » comme séparateur entre les valeurs d'une variable.

XI. Contrôle de processus :

La commande « **ps** » liste les processus de la même manière que la commande « **ls** » liste les fichiers. Par défaut, il affiche uniquement des informations sur les processus lancés à partir de votre terminal. Seulement l'ID de processus, Terminal, le temps écoulé et les options des commandes sont affichées.

L'option « **-e** » affiche des informations sur tous les processus en cours d'exécution dans le système.

L'option « **-f** », en plus de l'information fournie par défaut par « **ps** », affiche le nom de l'utilisateur, PPID, l'heure de début pour chaque processus (c'est à dire une liste complète).

1. Processus au premier plan :

Il prend le contrôle total sur le terminal pendant qu'il est en cours d'exécution

Exemple :

```
ls -R / > bigfile
```

2. Processus en arrière-plan :

Il s'exécute sans interaction avec le SHELL.

Exemple :

```
ls -R / > bigfile &
```

3. Interrompre un processus :

1.1 Processus au premier plan :

Les processus au premier plan interagissent avec le terminal. Ce type de processus peut être arrêté par un signal d'interruption en appuyant sur <**CTRL-C**>. Parfois, le <**CTRL-C**> ne fonctionne pas. Dans ce cas, vous devez utiliser la commande « **kill** » pour arrêter le processus.

1.2 Processus en arrière-plan :

Les processus en arrière-plan ne sont pas en interaction avec le terminal et doivent être arrêtés à l'aide de la commande « **kill** ».

1.3 La commande « kill » :

La commande « **kill** » est utilisé pour communiquer un changement d'état d'une commande. Le nom de la commande « **kill** » est trompeur parce que de nombreux signaux n'arrêtent pas les processus.

La commande peut être utilisée pour arrêter l'exécution d'un processus, mais peut aussi être utilisé pour transmettre d'autres changements d'état de processus.

« **kill** » utilise des signaux pour communiquer avec le processus. Si aucun signal n'est spécifié, la commande « **kill** » envoie un signal par défaut « 15 » dire au processus de se terminer.

NB :

Un utilisateur « **root** » peut arrêter tout processus avec la commande « **kill** ». Si vous n'êtes pas un utilisateur « **root** », vous devez avoir initié le processus pour que vous puissiez l'arrêter.

1.4 Les signaux de la commande kill:

Signal	Signification
1	Déconnexion sur terminal de contrôle (déconnexion quand le processus est en cours d'exécution)
2	Interrompre l'exécution (CTRL+C)
3	Quitter (CTRL+\)
9	Forcer l'arrêt d'un processus (ne peut pas être ignoré)
15	Par défaut, envoyer la demande d'arrêt au processus (arrêt normal)

4. Exécution des processus long :

La commande « **nohup** » ou « **no hangup** » va prendre le contrôle d'un processus en arrière-plan une fois que le processus a été invoqué. Il informe le processus d'ignorer les signaux 1 et 3 (raccrocher et quitter). Cela permettra la continuité d'exécution du processus si vous vous déconnectez du système.

Exemple :

```
nohup ls -R / > out 2> err.file &
```