



Interface en ligne de commande

Chapitre 1 : Introduction Générale	2
1. Système d'exploitation	2
2. Unix ?	2
3. Qu'est-ce que le GNU ?	3
4. Qu'est-ce qu'un logiciel libre ?	3
5. Linux	4
6. Structure d'un système Linux	4
7. Ouverture et fermeture de sécession	5
Chapitre II : les systèmes de gestion des fichiers	6
1. Les types de fichiers	6
2. La permission	7
3. Gestion basique de fichiers	8
Chapitre III : Programmation Scripts BASH	9
1. Shell	9
2. Les variables BASH	12
3. La commande test	14
Chapitre IV : Commandes GNU & Unix	17
1. Commandes de base	17
2. Pages de manuel	19
3. utiliser une variable	20
4. Les commandes générales	20

Chapitre 1 : Introduction Générale

Linux ou GNU/Linux est une famille de systèmes d'exploitation open source de type Unix fondé sur le noyau Linux, créé en 1991 par Linus Torvalds. De nombreuses distributions GNU/Linux ont depuis vu le jour et constituent un important vecteur de popularisation du mouvement du logiciel libre.

Si à l'origine, Linux a été développé pour les ordinateurs compatibles PC, il n'a équipé qu'une très faible part des ordinateurs personnels. Mais le noyau Linux, accompagné ou non des logiciels GNU, est également utilisé par d'autres types de systèmes informatiques, notamment les serveurs, téléphones portables, systèmes embarqués ou encore superordinateurs. Le système d'exploitation pour téléphones portables Android utilise le noyau Linux et équipe aujourd'hui 85 % des tablettes tactiles et smartphones.

« Au sens strict, linux est le nom du noyau du système d'exploitation libre, multitâche, multiplateforme et multi-utilisateurs de type Unix créé par Linus Torvalds, souvent désigné comme le noyau linux ».

Linux ? Unix ? Logiciels libres ? Faisons donc un petit voyage dans l'historique des systèmes d'exploitation pour y voir un peu plus clair.

Aujourd'hui les ordinateurs les plus puissants de la terre tournent à peu près exclusivement sous Linux. D'après les statistiques en novembre 2016, 498 des 500 machines recensées par top500.org tournent sous linux, trois Unix et aucune sous Windows.

- L'infrastructure d'internet est assurée en grande partie par linux. Des serveurs d'entreprises comme Google, Facebook ou Amazon fonctionnent sous Linux.
- Les systèmes Unix embarqués sont omniprésents dans notre quotidien et font tourner à peu près tout, du modem routeur ADSL au téléviseur, du distributeur des billets de trains au système de navigation GPS.

1. Système d'exploitation

Un système d'exploitation est un ensemble de programmes chargé de faire l'interface entre l'utilisateur et le matériel. C'est-à-dire que quand un utilisateur tape une commande au niveau d'un logiciel (ou application), le logiciel interprète la commande, la transmet au système d'exploitation qui la transmet au matériel dans un format compréhensible.

2. Unix ?

A la fin des années 60, les constructeurs d'ordinateurs proposaient chacun de leur côté un système d'exploitation propre à leur machine et incompatible avec les autres. C'est dans ce contexte qu'aux Bel Labs de New York, une poignée des chercheurs (Dennis Ritchie, Ken Thompson) ont entrepris de réfléchir à un système d'exploitation idéal. Leurs réflexions les ont conduits à concevoir un produit qui a rapidement relégué ses concurrents au rang de curiosité et qui est encore abondamment utilisé de nos jours dans les applications industrielles, ou parfois domestiques : Unix.

La raison du rapide développement d'Unix fut la distribution de son code source aux universités américaines : chacune d'entre elles était libre de l'étudier et de le modifier ou de proposer des améliorations. Ces pratiques ayant malheureusement pour conséquence la réintroduction des incompatibilités entre les systèmes, il est apparu nécessaire de normaliser le comportement du système Unix. La norme POSIX était née.

3. Qu'est-ce que le GNU ?

Les systèmes Unix étaient encore, au début des années 1980, propriétaire : ils étaient proposés par des grands constructeurs pour leurs machines. Cette situation était frustrante pour les étudiants et techniciens qui ne pouvaient s'offrir une licence ou travailler avec toute la liberté qu'ils auraient souhaitée sur ces systèmes. Le besoin a pour ainsi dire suscité des initiatives alternatives, dont la première fut le lancement de GNU par Richard Stallman.

En 1984, Richard Matthew Stallman, chercheur en informatique du MIT quitte son poste et se consacre à l'écriture d'un système d'exploitation libre du nom de GNU. Il annonce l'année suivante la création de la Free Software Fondation (FSF, fondation du logiciel libre) afin de supporter ce projet.

L'objectif était titanesque : il s'agissait d'écrire un système Unix complet en repartant de zéro, de manière compatible avec les systèmes existant, et sous forme de logiciel libre. Développeur talentueux et émérite, ancien chercheur au laboratoire d'intelligence artificielle du MIT, l'une des universités les plus réputées des Etats unis d'Amérique, RMS a été rapidement rejoint par des collaborateurs et des volontaires du monde entier. Pièce après pièce l'édifice prenait forme.

Dans la seconde moitié des années 1980, le projet GNU progresse lentement, mais surement. En 1990, l'ensemble des composants est réalisé et il ne manque plus que le noyau du système. Or, le noyau ou le Kernel constitue la partie la plus importante du code, la pièce maîtresse, celle qui se situe le plus près du matériel et qui contrôle les fonctions élémentaires comme la gestion de la mémoire, le contrôle des périphériques et des processus. C'est ainsi en début janvier 1991, de l'autre côté de l'atlantique en Finlande que vient le salut.

4. Qu'est-ce qu'un logiciel libre ?

L'expression « logiciel libre » fait référence à la liberté et non pas au prix. Pour comprendre le concept, vous devez penser à la « liberté d'expression », pas à « l'entrée libre ».

L'expression « logiciel libre » fait référence à la liberté pour les utilisateurs d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel.

Selon Richard Stallman, un logiciel est libre s'il respecte les quatre conditions fondamentales suivantes :

- La liberté d'utiliser le logiciel ;
- La liberté de le copier ;
- La liberté d'en étudier le fonctionnement ;
- La liberté de le modifier et de redistribuer cette version modifiée.

Licence Générale GNU (ou GPL) que Stallman publie en 1989.

5. Linux

Au début des années 1990, tous les éléments du système GNU étaient prêts. Seul manquait le noyau, cœur assurant la liaison de l'ensemble, couche intermédiaire entre le matériel et les éléments du système. C'est ainsi, un étudiant finlandais qu'indisposait la faible disponibilité de l'ordinateur serveur Unix de l'université d'Helsinki, entreprit d'écrire un macronoyau Unix, gros programme regroupant toutes les fonctionnalités réparties en autant des composants sous Hurd (c'est un micronoyau entouré d'une horde de modules spécialisés chacun en un périphérique particulier). Linus Torvalds (c'est son nom) de l'autre côté de l'Atlantique en Finlande, le jeune étudiant en informatique décide d'investir dans du matériel informatique. Il commande en contractant une dette sur 3 ans pour acheter un ordinateur. Il commande un jeu de disquettes d'installation du système Minix, une variante pédagogique d'Unix développée par le professeur Andrew Tanenbaum, qui servira pour les cours de l'architecture des systèmes d'exploitation.

Linus Torvalds a lui aussi rapidement été rejoint et son projet initié en 1991 est rapidement devenu fonctionnel. Dès 1993, les premières solutions complètes intégrant le noyau linux et le système GNU, ainsi que quelques applicatifs, ont vu le jour. Elles n'ont depuis pas cessé de se développer en volume, qualité et base installé.

- Les familles de distribution Linux

Il existe deux familles de distributions Linux à savoir :

- DEBIAN (Ubuntu, Damn Small Linux, etc.) ;
- Red Hat (RHEL, Fedora, Centos, Mandrivas, etc...).

Pour leurs installations peu des ressources matérielles sont exigées :

- dual-boot (Debian/Fedora);
- sur machines virtuelles (virtual box).

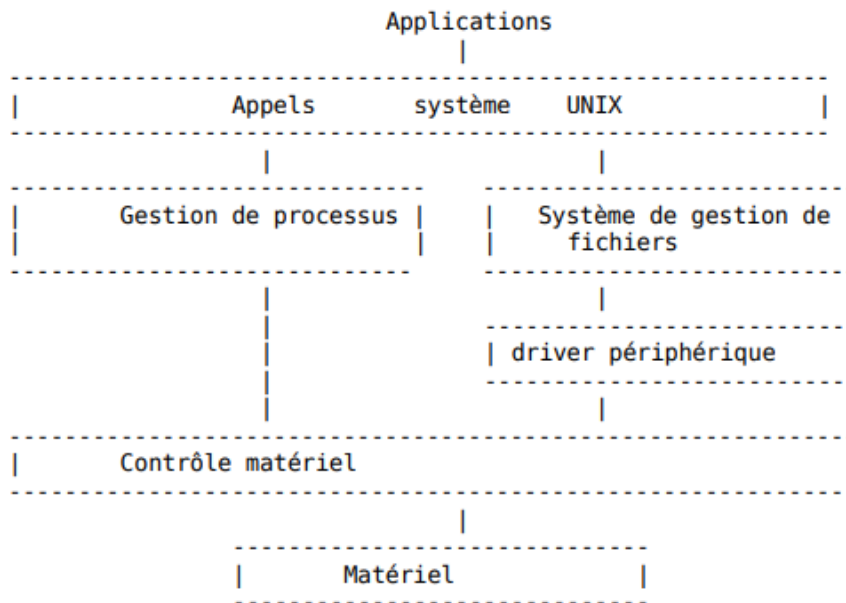
6. Structure d'un système Linux

Concrètement le système d'exploitation est lui aussi un ensemble de programme et de sous programmes regroupés dans ce qu'on appelle un noyau (Kernel en anglais).

Les process ne pouvaient pas accéder directement aux ressources matériels, en fait les process passent par le noyau pour y accéder, pour cela ils disposent d'un ensemble de commande appelées « appels systèmes » UNIX.

Ces appels systèmes commandent deux composantes principales du noyau, le gestionnaire de processus et le système de gestion de fichiers. Le premier a pour rôle de faire en sorte que les process s'exécutent et accèdent à la mémoire de manière équitable, on le nomme aussi **scheduler**. Le deuxième a pour rôle la gestion du système de fichiers, notamment pour ce qui concerne les droits d'accès.

Ce sont ces deux derniers composants du noyau qui accèdent directement au matériel.



7. Ouverture et fermeture de sécession

7.1. Ouverture de sécession

Avant de tenter une connexion, il faut d'abord vous assurer que vous avez été déclaré sur la machine, c'est-à-dire que vous possédiez un compte utilisateur caractérisé par un nom ou login et un mot de passe associé.

A la mise sous tension, apparaissent à l'écran toute une liste de termes plus ou moins barbares, vous pouvez ignorer tout ça. Au bout d'un certain temps apparaît enfin un message **login** : un curseur qui clignote. Le système attend que vous rentriez votre login. Rentrez votre login. Puis tapez **Entrer**, apparaît alors le message **Password**, tapez votre mot de passe, vous pouvez vous rendre compte que votre mot de passe n'apparaît pas en clair à l'écran, il est masqué pour des raisons de sécurité et vous ne voyez que le curseur qui clignote.

Une fois le login et le mot de passe saisi, deux possibilités peuvent s'offrir à vous, vous pouvez retrouver un écran noir, avec tout simplement un caractère du genre \$ ou > (appelé prompt) suivi du curseur qui clignote apparaît. Vous êtes dans un Shell prêt à taper des commandes. Par exemple, sous linux le prompt par défaut est le suivant :

```
[login@localhost login]$
```

Ou alors vous pouvez trouver un environnement fenêtre avec utilisation de la souris, où il vous sera possible de lancer un shell pour pouvoir taper des commandes linux.

7.2. Changement de mot de passe

En vous déclarant sur la machine, on vous a imposé un mot de passe, vous pouvez le changer, pour cela vous disposez de la commande **passwd**.

Si vous avez oublié votre mot de passe, vous devez vous adresser à l'administrateur du système (root) qui est le seul habilité à vous débloquent.

7.3. Fermeture de session

Quand on a fini d'utiliser le système, on doit se déconnecter ou fermer la session. Si vous êtes dans un environnement non graphique, il vous suffit au prompt de taper `logout`. Vous vous retrouvez alors avec le prompt de login, un autre utilisateur pourra alors utiliser la machine.

Dans un environnement graphique, vous avez une commande `exit`, ou `logout`, qui a strictement le même effet.

Vous devez veiller à vous déconnecter quand vous n'utilisez plus le système, pour des raisons de sécurité, mais aussi tout simplement pour libérer le poste de travail.

Chapitre II : les systèmes de gestion des fichiers

Le système n'impose aucune structure particulière aux fichiers sauf pour certains le concernant directement. Il ne fait aucune supposition quant à leur contenu et leur utilisation, cela ne regarde que les programmes qui construisent ou qui exploitent les fichiers. Chaque fichier est décrit par un enregistrement appelé **i-node** qui précise son type, sa taille, la date de création et de sa dernière modification, le numéro du propriétaire et du groupe, les permissions, les liens... les i-nodes sont rangés dans une table et chacun est identifié par un numéro.

1. Les types de fichiers

LINUX distingue quatre types de fichiers. Pourquoi autant ? Parce que pour linux tout est fichier. Par exemples, tous les périphériques sont des fichiers.

- Les fichiers ordinaires sont de type bloc. Leurs données sont lues et écrites par blocs.
- Les répertoires ou dossiers permettent d'organiser le système de fichiers en arborescence hiérarchique.
- Les liens (physique ou symbolique) sont des raccourcis qui permettent d'accéder à un fichier sous plusieurs noms.
- Enfin, on trouve des types de fichiers spéciaux, servant à communiquer avec les périphériques ou à synchroniser des données.

1.1. Fichiers et répertoires cachés

Linux utilise des fichiers et des répertoires que l'on dit cachés, en ce sens qu'ils ne sont normalement pas affichés dans un explorateur ou avec la commande `ls`.

Ces fichiers et ces répertoires sont les plus souvent des fichiers de configuration et n'ont pas vocation à être visualisés ou modifiés souvent par l'utilisateur : les cacher évite donc de polluer son environnement. Cependant, vous pouvez avoir besoin de les afficher, pour des raisons de maintenance.

Avec la commande `ls`, vous devez ajouter l'option `-a`.

Pour voir les fichiers cachés, placez-vous dans votre répertoire personnel (attention pas dans le répertoire `documents` si vous en avez un, mais bien dans `/home/utilisateur`), et validez l'affichage de ces fichiers.

Les noms de tous ces fichiers et répertoires ont une particularité commune : ils commencent par un point(.).

1.2. Les noms de fichiers

De nombreuses commandes du Shell ainsi que les programmes font référence à des noms de fichier pour trouver l'information désirée. Il est important de savoir écrire correctement un nom de fichier et en particulier dans les deux écritures référence : absolue et relative. La manière de nommer les noms de fichier utilise la structure arborescente des répertoires avec comme caractère séparateur le caractère « slash »/.

Il existe deux façons de spécifier l'emplacement d'un fichier :

- soit on indique son emplacement à partir de la racine de l'arborescence, c'est ce que l'on nomme le chemin absolu ;
- soit on indique son emplacement à partir de l'emplacement où l'on se trouve actuellement, c'est le chemin relatif.

1.2.1. Chemin absolu

Un chemin absolu débute par le caractère /, qui représente la racine du système de fichiers. Ainsi, `/usr/lib/locate/code` désignera le fichier code du répertoire locale, placé lui-même dans le répertoire lib, situé lui dans le répertoire usr, ce dernier étant dans la racine du système de fichiers.

1.2.2. Chemin relatif

Un chemin relatif exprime l'emplacement d'un fichier ou d'un répertoire à partir du répertoire courant (celui dans lequel on se trouve). Reprenons l'exemple précédant : nous sommes dans `/usr/lib`, soit dans le répertoire lib, situé immédiatement dans le répertoire usr, lui-même dans le répertoire racine.

Depuis cet emplacement, on pourra alors trouver le fichier `/usr/lib/locate/code` dont nous parlions ci-dessus, en se référant simplement à `locate/code`.

En ligne de commande, le répertoire courant est désigné simplement par un point «.», et le répertoire père par deux point «..». Il s'agit bien ici de chemin relatif, puisqu'on se réfère au répertoire dans lequel on se trouve, et non à la racine de l'arborescence.

2. La permission

Tout utilisateur d'un système UNIX possède un numéro d'utilisateur unique et un numéro de groupe commun à plusieurs utilisateurs. Les permissions des fichiers (lecture, écriture, exécution) sont définies relativement à trois domaines : le répertoire du fichier, le groupe comprenant les utilisateurs du même groupe que le propriétaire, et les autres. Les permissions sont listées dans l'ordre propriétaire, groupe et autres. Pour chaque domaine sont écrites la permission de lecture (r), écriture (w) et exécution (x). Lorsque la permission n'est pas accordée un tiret remplace la lettre correspondante. Par exemple `'rwxr-x-x'` indique un fichier sur lequel le propriétaire a tous les droits, le groupe ceux de lire et d'exécuter, et les autres celui d'exécuter.

Bien évidemment, le propriétaire a seul le pouvoir de modifier les permissions.

Chaque programme ou logiciel attribue des permissions septiques lors de la création de fichiers.

3. Gestion basique de fichiers

3.1. Nommage

- Maximum 255 caractères.
- Sensible à la case.
- Se limiter aux caractères spéciaux suivants :
 - point (.) ;
 - délimiteurs (- et _) ;
 - espace avec échappement ou quotes ;
 - tilde en fin de fichier (indique un fichier de sauvegarde automatique).

3.2. Chemin d'accès

- Le délimiteur est le slash (/).
- Racine unique : le dossier /.
- Deux types de chemins :
 - absolu.
 - relatif.

Chapitre III : Programmation Scripts BASH

1. Shell

Le Shell est l'interprète des commandes Unix, c'est un programme qui dialogue avec tout utilisateur. Dès que la procédure de connexion (login) est terminée, il prend le contrôle, lit les commandes que vous tapez, les interprète et lance les actions adéquates.

Avec les systèmes Unix, l'utilisateur trouvera de nombreux outils pour contrôler complètement sa machine et ses logiciels. Le Shell, ou interpréteur des commandes, est l'un des programmes les plus utilisés sous Unix. C'est en effet lui qui va permettre de communiquer avec son système.

1.1. Les différents Shells Unix

Lorsque les premiers systèmes commençaient à voir le jour grâce aux travaux conjugués de Dennis Ritchie et Ken Thompson, le besoin d'améliorer les interpréteurs de commandes de l'époque se fit ressentir. C'est dans cette optique que le Shell sh fut créé en 1974 par Steve Bourne. Celui qui implémentait des fonctions plus performantes et introduisait ainsi un véritable langage de programmation scripté. Par la suite, de nombreux autres shells virent le jour, chacun essayant d'améliorer le confort d'utilisation et l'efficacité de sa programmation.

Parmi les shells les plus connus :

- Bash (le plus répandu) ;
- Bourne shell (tout premier shell Unix) ;
- Tcsh (second shell de référence et le plus moderne) ;
- C Shell (plus proche de C) ;
- Korn Shell (AIX&HP/UX) développé par David Korn ;
- Z Shell (plus innovant et synthèse de tous les autres).

1.2. Choisir un shell

Pour les débutants sur Linux c'est le Bash qui est conseillé, très familier d'autres systèmes Unix.

Ses avantages :

- Démarrage en mode texte.
- Shell accessible directement après authentification.

1.3. Programmation Shell

Un script bash est un fichier de type texte contenant une suite de commandes shell, exécutable par l'interpréteur (ici le programme `/bin/bash`), comme une commande unique. Un script peut être lancé en ligne de commande, comme dans un autre script. Mais il s'agit bien plus qu'un simple enchaînement de commande : on peut définir des variables et utiliser des structures de contrôle, ce qui lui confère le statut de langage de programmation interprété et complet.

Le langage bash gère notamment :

- la gestion des entrées-sorties et de leur redirection
- des variables définies par le programmeur et des variables systèmes
- le passage de paramètres
- des structures conditionnelles et itératives
- des fonctions internes

1.3.1. Saisie du script

Les lignes commençant par le caractère dièse # sont des commentaires. Le script doit débuter par l'indication de son interpréteur écrite sur la première ligne :

!/bin/bash.

Exemple :

```
# !/bin/bash
# script bonjour
# affiche un salut à l'utilisateur qui l'a lancé
# l'option -n empêche le passage à la ligne
# le ; sert de séparateur des commandes sur la ligne
echo -n "Nous sommes le " ; date
# recherche de $USER en début de ligne dans le fichier passwd
```

1.3.2. Exécution du script

Il est indispensable que le fichier script ait la permission **x** (soit exécutable).

Lui accorder cette permission pour tous ses utilisateurs avec **chmod** :

Exemple : `$chmod +x bonjour.sh.`

Pour lancer l'exécution du script, taper **./bonjour.sh**, **./** indiquant le chemin, ici le répertoire courant. Ou bien indiquer le chemin absolu à partir de la racine. Ceci dans le cas où le répertoire contenant le script n'est pas listé dans le PATH.

Si les scripts personnels sont systématiquement stockés dans un répertoire précis, par exemple **/home/bin**, on peut ajouter ce chemin dans le PATH.

Pour cela, il suffit d'ajouter la ligne suivante dans **/etc/skel/.bash_profile**, qui est recopié dans chaque répertoire dont le rôle est d'affiner le profil personnel du shell de chaque utilisateur.

```
# bash_profile
...
...
```

```
...  
#user specific environment and statup programs  
PATH=$PATH :$HOME/bin
```

Mais on peut plus simplement s'initier au langage Bash, directement en dialoguant avec l'interpréteur. Si on entre une instruction incomplète en ligne de commande, l'interpréteur passe à la ligne suivante en affichant le prompt `>` et attend la suite de l'instruction (pour quitter Ctrl-C).

1.3.3. Entrées-sorties

Ce sont les voies de communication entre le programme bash et la console :

echo, affiche son argument texte entre guillemets sur la sortie standard, c'est-à-dire sur l'écran.

La validation d'une commande echo provoque un saut de ligne.

```
echo "Bonjour à tous !"
```

On peut insérer les caractères spéciaux habituels, qui seront interprétés seulement si l'option -e suit echo.

`\n` (saut ligne), `\b` retour arrière), `\t` (tabulation), `\a` (alarme), `\c` (fin sans saut de ligne)

```
echo "Bonjour \n à tous !"  
echo -e "Bonjour \n à tous !"  
echo -e "Bonjour \n à toutes \net à tous !\c"
```

La sortie :

La sortie est une notion très importante de scripting Bash, étant donné que les différentes commandes du système doivent envoyer 0 pour mentionner que leur exécution est correcte.

La sortie :

- Met fin à l'exécution du script ;
- Renvoie la valeur donnée ;
- 0 = bonne exécution ou true.

read,

Permet l'affectation directe par lecture de la valeur, saisie sur l'entrée standard au clavier

`read var1 var2 ...` attend la saisie au clavier d'une liste de valeurs pour les affecter, après la validation globale, respectivement aux variables `var1`, `var2 ...`

```
echo "Donnez votre prénom et votre nom"  
read prenom nom
```

```
echo "Bonjour $prenom $nom"
```

1.3.4. Instruction CLEAR pour effacer l'écran

Pour effacer le texte de la console on utilise l'instruction **clear** (CLear the Screen)

- Insérer en début de fichier elle aura pour effet d'effacer le texte de la console qui s'y trouvait avant l'exécution de votre fichier BATCH
- Insérer derrière une commande qui produit une sortie écran importante elle videra l'écran

2. Les variables BASH

2.1. Variables programmeur

De façon générale, elles sont de type texte.

On distingue les variables définies par le programmeur et les variables systèmes

Syntaxe : **variable=valeur**

Attention ! Le signe = NE DOIT PAS être entouré d'espace(s).

On peut initialiser une variable à une chaîne vide :

chaîne_vide=

Si la valeur est une chaîne avec des espaces ou des caractères spéciaux, l'entourer de " " ou de ' '.

Le caractère \ permet de masquer le sens d'un caractère spécial comme " ou '.

```
chaîne= " Bonjour à tous "  
echo $chaîne
```

Référence à la valeur d'une variable : faire précéder son nom du symbole \$.

Pour afficher toutes les variables : **set**.

Pour empêcher la modification d'une variable, invoquer la commande **readonly**.

2.1.1. Substitution de variable :

Si une chaîne contient la référence à une variable, le shell doit d'abord remplacer cette référence par sa valeur avant d'interpréter la phrase globalement. Cela est effectué par l'utilisation de " ", dans ce cas obligatoire à la place de ' '.

Exemples :

```
n=123 ;  
echo "la variable \n vaut $n"  
salut="bonjour à tous !"  
echo "Alors moi je dis : $salut"
```

```
echo 'Alors moi je dis : $salut'
echo "Alors moi je dis : \"$salut\" "
readonly salut
salut="bonjour à tous, sauf à toto"
echo "Alors moi je dis : $salut"
```

2.1.2. Variables exportées

Toute variable est définie dans un shell. Pour qu'elle devienne globale elle doit être exportée par la commande :

`export variable`

`export` → Pour obtenir la liste des variables exportées.

Opérateur {} dans les variables

Dans certains cas en programmation, on peut être amené à utiliser des noms de variables dans d'autres variables. Comme il n'y a pas de substitution automatique, la présence de {} force l'interprétation des variables incluses.

Voici un exemple :

```
user="/home/stage"
echo $user
u1=$user1
echo $u1 → ce n'est pas le résultat escompté !
u1=${user}1
echo $u1
```

2.1.3. Variables d'environnement

Ce sont les variables systèmes dont la liste est consultable par la commande `env` | `less` ou `printenv`

Les plus utiles sont `$HOME`, `$PATH`, `$USER`, `$PS1`, `$SHELL`, `$ENV`, `$PWD`, `$HOSTNAME`,...

Exemples :

```
moi=Toto
p="Je m'appelle $moi"
```

```
echo Aujourd\'hui, quel jour sommes-nous ? ; read jour
```

```
echo Aujourd\'hui $jour, $moi sous le nom $USER, est connecté à la station $HOSTNAME
```

2.1.4. Variables prédéfinies spéciales

Elles sont gérées par le système et s\'avèrent très utiles dans les scripts.

Bien entendu, elles ne sont accessibles qu\'en lecture.

Ces variables sont automatiquement affectées lors d\'un appel de script suivi d\'une liste de paramètres. Leurs valeurs sont récupérables dans \$1, \$2 ...\$9

\$?	C\'est la valeur de sortie de la dernière commande. Elle vaut 0 si la commande s\'est déroulée sans problème.
\$0	Cette variable contient le nom du script
\$1 à \$9	Les (éventuels) premiers arguments passés à l\'appel du script
\$#	Le nombre d\'arguments passés au script
\$*	La liste des arguments à partir de \$1
\$\$	Le n° PID du processus courant
\$!	Le n° PID du processus fils

3. La commande test

3.1. Généralités

Comme son nom l\'indique, elle sert à vérifier des conditions. Ces conditions portent sur des fichiers (le plus souvent), ou des chaînes ou une expression numérique.

Cette commande courante sert donc à prendre des (bonnes) décisions, d\'où son utilisation comme condition dans les structures conditionnelles : if.. then ..else.

Syntaxe

test expression

[expression] attention aux espaces autour de expression !

3.2. Valeur de retour

Rappels : On sait que toute commande retourne une valeur finale au shell : 0 pour lui indiquer si elle s\'est déroulée normalement ou un autre nombre si une erreur s\'est produite.

Cette valeur numérique est stockée dans la variable spéciale **\$?**

La commande test, de même, retourne **0** si la condition est considérée comme vraie, une valeur différente de **0** sinon pour signifier qu\'elle est fausse.

3.3. Tester un fichier

Elle admet 2 syntaxes (la seconde est la plus utilisée) :

test option fichier

[option fichier]

Tableau des principales options

Option	signification quant au fichier
-e	il existe
-f	c'est un fichier normal
-d	c'est un répertoire
-r -w -x	il est lisible modifiable exécutable
-s	il n'est pas vide

Exemples :

```
[ -s $1 ]      vrai (renvoie 0) si le fichier passé en argument n'est pas vide
[ $# = 0 ]     le nombre d'arguments est 0
[ -w fichier ] le fichier est-il modifiable ?

[toto@p00]$ [ -r "/etc/passwd" ]      toto peut-il lire le fichier /etc/passwd ?
[toto@p00]$ echo $ ?  -> 0 (vrai)
[toto@p00]$ [ -r "/etc/shadow" ]      toto peut-il lire le fichier /etc/shadow ?
[toto@p00]$ echo $ ?  -> 1 (faux)
[toto@p00]$ [ -r "/etc/shadow" ] || echo "lecture du fichier interdite"
```

Expression mathématique :

- Calculer et afficher le resultat sur la sortie standard

Expr expression

- Exécuter une opération

let expression

Ou

((Expression))

- Peut être précédé de \$ pour enregistrer dans une variable ou afficher
\$expr 2+5

- Opération

```
$ rslt=$((2+3))  
$echo $rslt  
5
```

Composition :

- ❖ Séquentielle : cmd1 ; cmd ;
- ❖ Parallèle : cmd1 & cmd2 ;
- ❖ Sur erreur (ou) : cmd1 || cmd2 ;
- ❖ Sur succès (et) : cmd1 && cmd2.

Chapitre IV : Commandes GNU & Unix

Une commande est une instruction qu'un utilisateur envoie au système d'exploitation de son ordinateur pour lui faire exécuter une tâche. Il peut s'agir de manipuler des fichiers, d'accéder à des répertoires, de modifier des droits d'accès, etc. Du fait de la complexité des systèmes d'exploitation, il en existe un très grand nombre, et les actions précises de chacune d'elles sont de plus conditionnées par un jeu plus ou moins volumineux d'options et de paramètres.

Elles constituent ainsi un outil extrêmement puissant, mais encore faut-il les connaître, et parfaitement comprendre leurs actions !

Ces commandes peuvent être déclenchées soit indirectement par le biais d'outils graphiques (outils de l'environnement) soit directement par le biais de lignes de commandes saisies sur un terminal ou une console dans un langage extrêmement concis pour en simplifier la frappe, au risque de les faire apparaître comme extrêmement mystérieuses aux néophytes voire même aux utilisateurs aguerris.

Dans cette partie des Commandes GNU & Unix, vous trouverez des commandes Linux essentielles à connaître qui sont utilisées pour contrôler le terminal. Elles vous apprendront à comment effacer la zone de visualisation du terminal, récupérer des entrées de terminal précédentes depuis l'historique, copier, déplacer, supprimer et ou terminer la session du terminal.

1. Commandes de base

Commande	Description
Clear	Vider le terminal Utilisez la commande <i>clear</i> pour effacer le contenu de l'écran. <i>clear</i> Vous obtiendrez un terminal vide avec une invite de commande (prompt). Au lieu d'utiliser cette commande, vous pouvez également vider votre terminal à l'aide de la combinaison de touches [CTRL] + [L].
Exit	Fin de session La commande <i>exit</i> termine la session en cours et ferme le terminal. <i>exit</i> Vous pouvez également utiliser la combinaison de touches [CTRL] + [D].
Help	Afficher une liste des commandes shell Utilisez la commande <i>help</i> pour afficher une liste de toutes les commandes Shell intégrées.

	<i>help</i> Utilisez <i>help</i> directement avec une commande shell pour afficher une courte description de la commande correspondante. <i>help COMMANDE</i>
--	---

Ligne de commande

Pour travailler en ligne de commande, vous devez tout d'abord vous retrouver face à une « invite de commande » (command prompt), c'est-à-dire quelque chose qui ressemble vaguement à ceci.

enastic@amdjarass:~\$

- tilde (~): c'est un symbole qui désigne votre répertoire d'utilisateur.
- Le dollar \$: signifie par convention qu'il s'agit d'un utilisateur.
- Le dièse # : il s'agit de l'utilisateur root.

Principe générale des commandes

La totalité des commandes prédéfinies (et la plupart du temps des programmes) suivent les conventions suivantes :

- Le nom d'une commande est suivi par un nombre quelconque d'options de la forme :
 - – caractère → option qui n'exige aucune valeur ;
 - – caractère valeur → option qui demande une valeur ;
 - Si aucune option ne comporte de valeurs, on doit les regrouper
 - L'ordre des noms de fichier dépend de la nature de la commande.

Utiliser l'aide

Toutes les commandes (à très peu d'exceptions près) disposent d'un véritable manuel en ligne, que l'on peut afficher grâce à man suivie du nom de la commande sur laquelle on souhaite se renseigner.

```
| Man nom_commande
```

Exemple : les deux commandes ci-dessous permettent d'afficher respectivement la page de manuel pour la commande man elle-même et la page de manuel de la commande cp.

```
| $ man man  
| $ man cp
```

Aide interne à une commande

L'aide interne à une commande n'est pas du tout standard c'est-à-dire que chaque commande pourra intégrer ou non. En général, l'on peut l'afficher avec l'une des options suivantes :

```
|| • - h
```

- - - help.
- -h.

Par exemple l'option -h de man \$ man -h vous permet d'afficher une aide extrêmement sommaire.

2. Pages de manuel




Sous linux les pages de manuel sont triées par catégorie et nous avons 09 categories à savoir :

1. Programmes exécutable ou commande de l'interpréteur de commande (Shell) ;
2. Appels système (fonctions fournies par le noyau) ;
3. Appels de bibliothèque (fonction fournies par les bibliothèques des programmes) ;
4. Fichiers spéciaux (situés généralement dans /dev) ;
5. Formats des fichiers et convention. Par exemple /etc/passwd ;
6. Jeux ;
7. Divers (y compris les macros paquets et les conventions), par exemple man (7), groff (7) ;
8. Commandes de gestion du système (généralement réservées aux super utilisateurs)
9. Sous-programmes du noyau (hors standard).

a) Info

La commande info est beaucoup plus complexe que la commande man. Elle amène à une véritable documentation. Il y a différentes pages au sein de info chacune sont rassemblées au sein des rubriques et les rubriques vont des thèmes plus général au plus particulier, cet également des documents en HyperText ce qui veut dire qu'il y a des liens qu'on peut suivre d'un document à un autre.

En résumé :

- Ensemble des pages hiérarchisées :
 -  Rubriques ;
 -  Du plus général au plus particulier.
- Liens :
 -  Marqués d'un astérisque (*).

Exemple : \$ info /*info tout seul ramène à la racine d'info */
\$ info cp

Info navigation

q	→ Quitter
espace	→ Défilement vers le bas
return	→ défilement vers le haut
b	→ début du nœud (beginning)
e	→ fin du nœud (end)
Tab	→ aller au lien suivant
Entrée	→ suivre le lien

n → nœud suivant (**n**ext)
p → nœud précédent
u → nœud de niveau supérieur (**u**p)
l (L min) → retour à la page précédemment affichée (**l**ast)

b) Les variables

- Permet d'associer une clé (son nom) à une valeur
 - ✚ En mémoire
 - ✚ Exemple : la variable « message » contenant la chaîne de caractère « bonjour ».
- Avec BASH
 - ✚ Aucun typage
 - ✚ Toute valeur de variable est considérée comme une chaîne de caractère.

Déclarer une variable

- Sans typage, aucun besoin de déclaration spécifique
- Une simple affectation suffit :
 - Signal égal ;
 - Sans espace.
- Exemple : \$mavriable = "Bonjour".
- Attention : le nom d'une variable est sensible à la case.
 - C'est-à-dire « mavvariable » n'est pas la même variable que « Mavvariable ».

3. utiliser une variable

- Nécessite le caractère \$ devant le nom.
- Remplacer cette référence par la variable, exemple : \$ echo \$mavvariable affiche Bonjour.
- Par défaut la portée d'une variable : locales :
 - ✓ n'est définie et disponible que par l'instance de bash.
 - ✓ n'est pas accessible par les autres programmes.

a) Les commandes générales

Echo [-n] argument...argument

Affiche les arguments sur la sortie standard, utile pour afficher le contenu de certaines variables de l'environnement.



```
% echo Bonjour
```

```
Bonjour
```

```
% echo $home (homme est une variable d'environnement qui donne /usr/locate le répertoire de connexion)
```

Le \$ indique que nous voulons l'affichage de la valeur de la variable HOME.

Cat fichier...fichier

Cette commande est très utile pour concaténer une série de fichiers en un seul fichier.

```
% cat fichier1 fichier2 fichier3 > fichiergeneral
```

(Met dans le fichiergeneral le contenu du fichier1, puis fichier2 et enfin celui du fichier3)

```
% cat fichier4 fichier5 >> fichiergeneral
```

(ajoute au fichiergeneral le contenu du fichier4, puis celui du fichier5)

hostname

Ecrit sur la sortie standard le nom de la machine utilisée.

more fichier...fichier

Affiche le contenu des fichiers indiqués. L'utilisateur peut taper :

- Un blanc pour l'affichage de la page suivante ;
- Un retour chariot (ou entrée) pour afficher une ligne supplémentaire ;
- Q pour quitter le programme more.

whoami

Affiche le nom de l'utilisateur (nom du « user » lors de la connexion).

Commandes de manipulation des fichiers

a) Les répertoires

pwd [-lp]

« Print working directory » affiche le répertoire courant.

- -l suivre les liens symboliques.
- -p ne pas suivre les liens symboliques.

cd

« change directory » permet de changer de répertoire.

~	Répertoire personnel
.	Répertoire courant

..	Répertoire courant du répertoire courant
-	Dernier répertoire dans lequel nous nous situons

```
mkdir chemin...chemin
```

Création des dossiers indiqués.

- `mkdir -p` pour créer plusieurs dossiers.
Exemple : `$ mkdir -p mon dossier1/mon dossier2/mon dossier4`

i. les autres commandes essentielles

```
cp [options] fichier1 fichier2  
cp [options] fichier...fichier repertoire
```

Cette commande permet de copier des fichiers. La première forme permet de copier le fichier1 sur le fichier2 (si fichier2 existe déjà, il est écrasé). Dans la deuxième forme, les fichiers sont copiés dans le répertoire indiqué, chacun conservant son nom.

Les deux options suivantes sont reconnues :

- i demande de confirmation pour chaque copier ;
- r si l'une des référence source est répertoire, la recopie est récursive pour cette référence.

```
mv fichier1 fichier2  
mv fichier...fichier repertoire
```

Cette commande permet de déplacer ou de renommer un fichier.

```
rm [option] fichier1 fichier2
```

Supprime les fichiers et les répertoires.

Deux options sont très utiles :

- i demande de confirmation avant la suppression.
- r traite récursivement tous les sous-répertoires.

NB : l'option de récursivité avec des noms génériques est utile, mais aussi très dangereuse. Ne taper jamais la commande « `rm -r *` » sans savoir votre répertoire de travail.

Lorsque vous vous connectez, vous êtes dans un répertoire, la plupart du temps celui-ci contient des fichiers de configuration et de définition de certaines variables. Leurs noms commencent tous par un point « `.login` », « `.cshrc` », une commande du style « `rm *` » dans ce répertoire les supprimeraient et à la prochaine tentative de connexion, vous auriez de sérieux problèmes.

```
chmod modefichier...fichier
```

Permet de modifier les permissions de fichiers dont vous êtes le propriétaire. Le mode peut être exprimé de manière absolue ou de manière relative. Cette dernière est la plus simple à exploiter car seuls les attributs spécifiés sont changés :

- une combinaison des lettres u (propriétaire), g (groupe), o (autres). L'absence de paramètres est équivalente à ugo (c'est-à-dire tous) ;

- le signe + pour ajouter, - pour enlever et = pour donner ce droit uniquement ;
 - une combinaison des lettres r (lecture), w (écriture), x (eXécution).
- ii. les commandes externes

Afficher le contenu d'un répertoire avec ls

La commande ls (comme list) affiche la liste des fichiers d'un répertoire. Invoquée sans arguments, elle montre le contenu du répertoire courant :

```
enastic@amdjarass:~$ ls  
Bureau Documents Modèles Musique Public Téléchargements Images
```

Comment lire ce résultat ? La commande ls nous a retourné les éléments situés dans notre répertoire d'utilisateur.

Le répertoire courant est celui dans lequel vous vous retrouvez au moment où vous saisissez la commande.

Pour afficher le contenu de la racine du système de fichiers, saisissez ceci :

```
enastic@amdjarass:/$ ls  
bin dev home cdrom lost+found mnt proc run srv tmp var boot etc lib
```

Et pour voir ce qu'il y'a dans /usr, il suffit d'invoquer la commande suivante :

```
enastic@amdjarass:~$ ls /usr
```