Université Ibn Tofail
Faculté des Sciences
Département Informatique

Système d'Exploitation I

Filière SMI
Semestre 3

Auteur : Salma AZZOUZI
Année Universitaire 2015-2016

Sommaire

Chapitre 1 : Notions sur UNIX

1. Introduction

Pour qu'un ordinateur soit capable de faire fonctionner un programme informatique (appelé parfois application ou logiciel), la machine doit être en mesure d'effectuer un certain nombre d'opérations préparatoires afin d'assurer les échanges entre le processeur, la mémoire, et les ressources physiques (périphériques).

Un **système d'exploitation**, est chargé d'assurer la liaison entre les ressources matérielles, l'utilisateur et les applications. Ainsi lorsqu'un programme désire accéder à une ressource matérielle, il ne lui est pas nécessaire d'envoyer des informations spécifiques au périphérique, il lui suffit d'envoyer les informations au système d'exploitation, qui se charge de les transmettre au périphérique concerné via son pilote.

Le système **Unix** est un **système d'exploitation multi-utilisateurs**, **multi-tâches,** ce qui signifie qu'il permet à un ordinateur **mono** ou **multi-processeurs** de faire exécuter simultanément plusieurs programmes par un ou plusieurs utilisateurs. Il possède un ou plusieurs interpréteurs de commandes (shell) ainsi qu'un grand nombre de commandes et de nombreux utilitaires (assembleur, compilateurs pour de nombreux langages, traitements de texte, messagerie électronique, ...). De plus il possède une grande portabilité, ce qui signifie qu'il est possible de mettre en œuvre un système Unix sur la quasi-totalité des plates-formes matérielles.

De nos jours les systèmes Unix sont très présents dans les milieux professionnels et universitaires grâce à leur grande stabilité, leur niveau de sécurité élevé et le respect des grands standards, notamment en matière de réseau.

Le but de ce cours est d'abord de donner un aperçu général du fonctionnement du système UNIX et de se familiariser avec ses commandes de bases : manipulation sous shell des fichiers et processus. Nous nous sommes efforcés de ne décrire que des commandes standards, qui devraient fonctionner sous toutes les versions d'UNIX et pour tous les utilisateurs.

2. Historique

- **1965** : Bell Laboratoies, General Electric et le M.I.T. (filiale d'une société américaine de télécommunications) lancent le projet MULTICS qui vise à créer un système d'exploitation ***multitâches*** et ***multi-utilisateurs*** en ***temps partagé***, implantable sur

tout ordinateur, assurant une portabilité des applications . Les chefs de projets étaient Ken THOMPSON et Dennis RITCHIE de Bell Laboratories .

- **1970 :** Apparition d'une première version de MULTICS en PL/1 .
- **1971** : Réécriture de MULTICS en Assembleur. En cette année Ken crée le langage B.
- **1973** : Dennis RITCHIE et Brian KERNIGHAN crée le langage C issu d'améliorations de B, en vue d'une réécriture d'UNIX, nom qui a succédé à MULTICS (*Uniplexed Information and Computer Service*).Il a été écrit UNICS en premier lieu .
- **1974-1977 :** Le code source d'Unix est distribué librement aux universités. En conséquence, UNIX a gagné la faveur de la communauté scientifique et universitaire. Il a été ainsi à la base des systèmes d'exploitation des principales universités. l'Université de Berkeley devient un centre moteur du développement UNIX. Ses versions portent le signe BSD

Depuis, plus d'une vingtaine de versions d'UNIX ont vu le jour. Les quatre versions principales sont BSD, SYSTEM V, POSIX (fédération de BSD et SYSTEM V) et LINUX proposé par Linus TORVALDS en 1991(un sosie du système d'exploitation Unix écrit entièrement de A à Z).

3. Architecture du système Unix

3.1. L'Architecture du système :

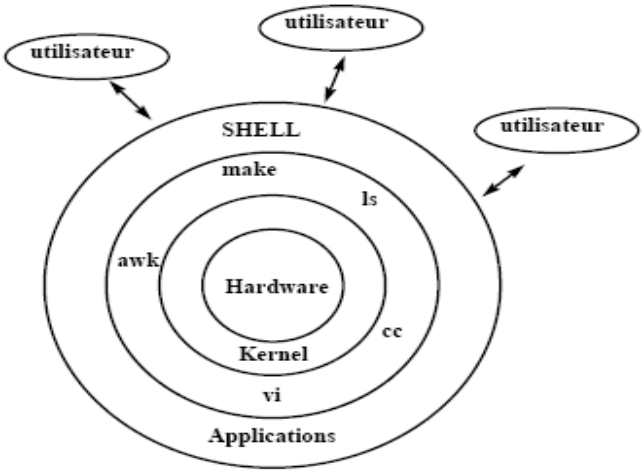

Figure 1 : L'architecture du système Unix

Comme le montre la Figure1, l'architecture globale d'UNIX est une architecture par couches (coquilles) successives. Les utilisateurs communiquent avec la couche la plus évoluée (la couche des applications). Le programmeur lui va pouvoir en fonction de ces besoins utiliser des couches de plus en plus profondes.

Chaque couche est construite pour pouvoir être utilisée sans connaître les couches inférieures (ni leur fonctionnement, ni leur interface). Ce principe d'encapsulation permet, alors, d'écrire des applications plus portables si elles sont écrites dans les couches hautes. Pour des applications ou le temps de calcul prime devant la portabilité, les couches basses seront utilisées.

3.2. L'Architecture du noyau

Le noyau offre les services de base ("interface système") pour construire les commandes, les outils, les applications :

- il gère les ressources matérielles (mémoire, unités d'E/S ...), les fichiers, l'allocation de temps CPU.
- il gère les processus.
- il gère la communication et la synchronisation

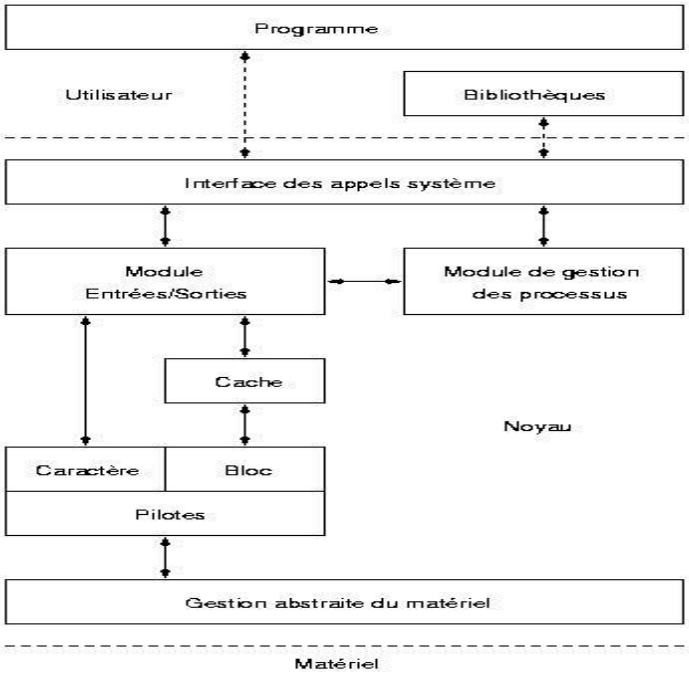

Figure 2 : Architecture du noyau

Pour faire appel aux services du système on passe généralement par des fonctions définies dans des bibliothèques qui fournissent une interface de programmation (API).Comme il est possible d'appeler directement le système en utilisant des interruptions logicielles prédéfinies (Figure 2).

Le noyau du système Unix est divisé en deux sous systèmes : le premier dont le rôle est de gérer tout ce qui concerne les entrées/sorties et le second dédié à la gestion des processus (ordonnancement, gestion mémoire, communication inter-processus, etc.).

Les entrées/sorties sont de deux types : en mode caractère pour des périphériques dont la nature même est de communiquer par caractères (les terminaux, les imprimantes et les réseaux échangent des caractères avec la mémoire centrale, sans tampon), et en mode bloc (les mémoires de masse (disque, disquettes) échangent des blocs d'information avec la mémoire centrale en utilisant des tampons.). Les entrées/sorties en mode caractère ne font pas appel au système de cache réservé aux blocs. Ce système est utilisé afin de minimiser les accès vers les périphériques concernés généralement bien plus lents que la mémoire.

4. Arborescence des fichiers Unix

Unix organise les données stockées sur les disques en fichiers (files) et répertoires (catalogues ou directories en anglais). Les fichiers contiennent les données elles-mêmes (images, textes, programmes. . .) et les répertoires permettent de les classer. Le classement prend pour analogie un arbre généalogique de paternité.

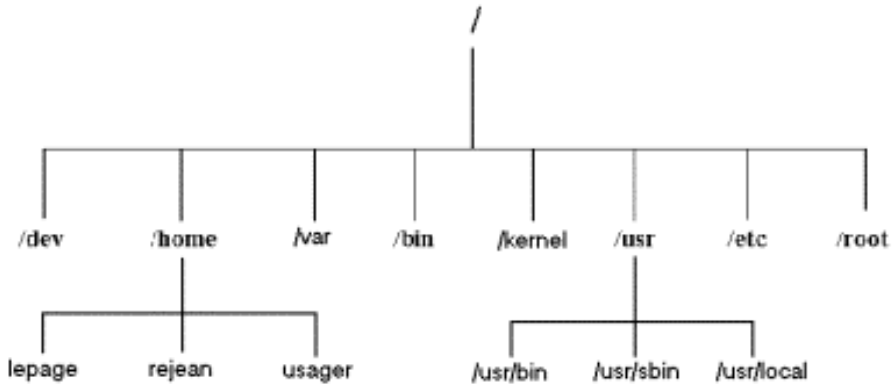

Figure 3 : Arborescence des fichiers d'UINX

Un répertoire P peut contenir plusieurs fichiers et plusieurs répertoires $F_1,\ldots, F_n$: on que dit les F_k sont les fils de P. Un même fichier/répertoire F ne peut être contenu que dans un seul répertoire P : on dit que P est le père de F. Le répertoire qui contient tous les autres fichiers/répertoires est appelé répertoire racine (***root***) de l'arborescence des fichiers. Par convention la racine est son propre père.

Chaque fichier/répertoire possède un nom (une suite de caractères limités à 14 caractères (/ est interdit et - ne peut être le premier caractère)).

Les fils d'un même père doivent avoir des noms distincts, mais deux fichiers/répertoires de pères différents peuvent avoir des noms identiques. Par convention la racine se nomme (/).

Il existe deux grands types de fichiers sous Unix :

- Les ***fichiers standards*** : les fichiers manipulés par les utilisateurs (les fichiers texte, les exécutables…..)
- Les fichiers spéciaux : manipulables que par l'intermédiaire du système (périphériques, mémoire …) pour communiquer avec un périphérique, on lit ou on écrit dans un fichier spécial représentant ce périphérique.

4.1. Chemin d'accès à un fichier

Le chemin d'accès à un fichier (ou un répertoire) est la description qui permet d'identifier le fichier (ou le répertoire) dans la hiérarchie du système. Le chemin d'accès correspond en une suite de noms de répertoires séparés par des caractères / (slash) et terminé par le nom du fichier ou du répertoire. Le caractère « / » marque la séparation entre catalogues lorsqu'on décrit le "chemin d'accès" à un fichier ou un catalogue.

Il y a deux façons de spécifier le chemin d'accès (***path***) à un fichier :

- soit on décrit le chemin à partir de la racine (/) c'est donc un chemin d'accès **absolu (référence absolue**). Le chemin d'accès absolu à un fichier est le même quel que soit l'endroit où on se trouve. On désigne ainsi un fichier/répertoire de façon unique, c'est-à-dire son nom précédé de toute la série de ses ancêtres depuis la racine. Chaque nom est séparé par autant de slash (/) que l'on veut.
- soit on décrit le chemin à partir du répertoire courant (catalogue courant), c'est un chemin **relatif** (il ne commence donc pas par un /). Le chemin d'accès relatif à un fichier sera différent en fonction du répertoire courant.

Exemple

- /home/Martin/Perso/mon_fichier.txt est une ***référence absolue*** (commence par /)
- ../../Martin/Perso/mon_fichier.txt est une ***référence relative*** (par rapport au répertoire courant).

4.2. Les répertoires

L'organisation traditionnelle du répertoire racine est décrite dans le tableau suivant :

Tableau 1 : Organisation du répertoire racine (root)

Répertoire	Contient
/bin	les fichiers exécutables nécessaires à l'initialisation
/boot	le noyau et les fichiers de démarrage
/dev	les fichiers spéciaux
/etc	les fichiers de configuration du système et certains scripts
/home	contient les répertoires personnels des utilisateurs. Par exemple, l'utilisateur toto a généralement pour répertoire `/home/toto`.
/lib	les librairies système et les modules
/lost+found	le stockage des fichiers retrouvés par fsck
/mnt	les points d'ancrage des systèmes extérieurs
/proc	un système de fichiers virtuels permettant l'accès aux variables du noyau
/root	le répertoire de base du super utilisateur
/sbin	les fichiers exécutables pour l'administration du système
/tmp	les fichiers temporaires
/usr	les programmes, les librairies et les fichiers accessibles pour l'utilisateur
/var	les données variables liées à la machine (spool, traces)

Ce système de fichiers peut résider sur différentes partitions, différents supports physiques ou sur d'autres machines sur le réseau. Ce découpage est complètement transparent pour les utilisateurs du système de fichiers. Les différentes parties peuvent être connectées au démarrage du système ou à la demande, en cours d'utilisation.

Dans la partie suivante, nous allons essayer de décrire quelques répertoires parmis les répertoires cités dans le tableau en haut :

4.2.1. Les répertoires home

Ce sont les répertoires qui regroupent les ” répertoires maisons” ***home directory*** des utilisateurs, c'est-à-dire les répertoires où ils sont chez “eux” et peuvent stocker leurs données personnelles. Le nom du home d'un utilisateur est généralement son login. Par sécurité, en dehors de son home, un utilisateur quelconque ne peut :

- Modifier n'importe quel fichier (droits d'écriture restreints sur les fichiers) : écriture interdite pour ne pas détruire le système ou les données d'un autre utilisateur par accident ou par malveillance.
- Ajouter ou supprimer un fichier dans n'importe quel répertoire.
- Lire n'importe quel fichier: lecture autorisée dans l'arborescence du système, sauf dans les homes où les utilisateurs fixent les droits comme ils souhaitent, et pour certains fichiers sensibles comme par exemple */etc/shadow* qui contient les mots de passe des utilisateurs.
- Exécuter n'importe quel fichier (droits d'exécution restreints sur les fichiers) : exécution interdite pour les fichiers de données (non destinées à être exécutées) : autorisées pour les exécutables, sauf les programmes d'administration du système comme ***useradd*** et ***usrdel*** pour ajouter ou supprimer un compte utilisateur.
- Aller n'importe où dans l'arborescence et en lister le contenu (resp. droits d'accès restreints en exécution et en lecture sur les r2pertoires) : Les utilisateurs aimant leurs intimités peuvent interdire aux autres l'accès à tout ou partie de leur compte.

4.2.2. Les répertoires bin

Les répertoires ***bin*** (comme ***binaire***) sont les répertoires destinés à contenir des fichiers exécutables. Si la plupart sont effectivement des binaires (suites de codes lisibles pour le processeur mais illisibles pour l'humain), d'autres sont des fichiers scripts (du texte lisible par l'humain) destinés à être interprétés par l'interpréteur adéquat :

- ***/bin*** : contient les binaires importants pour le fonctionnement du système (l'interpréteur de commande (ou shell) lui mˆeme, et les principales commandes pour la manipulation de fichiers comme cp (copy) pour dupliquer un fichier, rm (remove) pour effacer un fichier du disque. . . ..
- ***/usr/bin***, ***/usr/local/bin***, ***/usr/X/bin*** : des endroits où l'on installe habituellement les exécutables des applications utilisateurs.

  Exemple : dans /usr/local/bin on trouve
  - le programme de retouche de photos gimp (GNU Image Manipulation Programme),
  - le compilateur du langage C gcc (GNU C Compiler),
  - le visualiseur d'images xv (X11 viewer).
  - xterm, le programme qui ouvre un shell dans une fenêtre X11,
  - xclock un programme affichant l'heure sous la forme d'une horloge dans une fenêtre.

- ***/sbin, /usr/sbin*** : contient les programmes qui ne sont en principe utiles qu'aux administrateurs du système (ou sysadmin, super-user, root). La plupart de ces programmes nécessitent les privilèges du superuser pour être éxecutés, comme ***useradd*** et ***userdel*** qui permettent d'ajouter et de supprimer un compte utilisateur sur la machine.

4.2.3. Les répertoires lib

Les répertoires **lib** (comme library) sont les répertoires destinés aux fichiers bibliothèques : ce sont des blocs de code partagés par plusieurs codes binaires. Ceci a pour principe d'éviter que le même code soit dupliqué dans tous les binaires et on gagne donc de la place sur le disque. Ceci permet aussi de mettre à jour une bibliothèque en la remplaçant par une version plus récente (plus efficace ou contenant moins d'erreurs) et tous les binaires l'utilisant en profitent. On trouve ***/lib***, ***/usr/lib***, ***/usr/local/lib*** et ***/usr/X/lib***. Généralement, les noms de ces fichiers commencent par lib et se terminent par .so (shared object).

Exemple

- ***/lib/libc.so contient*** toutes les routines de base des binaires programmées en langage C (allocation mémoire, saisie au clavier, affichage sur le terminal,)
- ***/lib/libm.so contient*** toutes les routines mathématiques de base pour le C (racine carrée, algorithme, cosinus, . . .).

4.2.4. Les répertoires etc

Le répertoire ***etc*** contient les principaux fichiers de configuration et scripts d'initialisation du système. Ce sont généralement des fichiers texte.

- ***/etc/printers.conf*** : contient la liste des noms des imprimantes accessibles sur le réseau depuis la machine,
- ***/etc/passwd*** : contient la liste des utilisateurs pouvant se loger sur la machine. Chaque ligne construite sur le modèle :

```
login :pass :uid :gid :nom :home :shell
```

  - ***login*** : le login d'un utilisateur,
  - ***pass*** : le mot de passe de l'utilisateur,
  - ***uid*** : un entier unique identifiant l'utilisateur (une version numrique du login),
  - ***gid*** : est un nombre identifiant le groupe de l'utilisateur (par exemple Enseignants, Etudiants, Administrateur). Appartenir à tel ou tel groupe donne des privilèges sur la machine, notamment sur sur l'accés en lecture/écriture de certains fichiers/répertoires.
  - ***nom*** : c'est juste le nom de l'utilisateur dans la viecourante,
  - ***home*** : le chemin de son répertoire maison

- ***shell*** : son interpréteur de commande favori, utilisé par défaut par xterm.

Exemple

djelloul :x :158 :101 :Khalil DJELLOUL :/home2/djelloul :/bin/bash
godbert :x :133 :101 :GODBERT Elisabeth :/home2/godbert :/bin/bash

On voit que le champ pass est toujours vide. Par sécurité, les mots de passe sont cryptés dans le fichier inaccessible **/etc/shadow**.

4.2.5. Les répertoires man

Quand un programme est bien fait, il est également bien documenté. Le répertoire ***/usr/man*** (manual) regroupe les fichiers de documentation destinés à être lu par le programme ***/usr/bin/man***.

Exemple : pour utiliser la commande ***ls***, nous taperons la commande suivante :

```
man ls
```

De même pour utiliser le manuel lui-même, nous taperons :

```
man man
```

5. Les commandes Unix

Il existe deux environnements de travail différents sous les systèmes d'exploitations de la famille d'Unix (dans notre cours nous travaillerons toujours avec le système d'exploitation Linux) :

- le mode texte : les commandes sont saisies au clavier et le résultat s'affiche sous forme de lignes de texte sur un écran
- L'environnement ***Xwindow*** (l'interface des stations ***UNIX***) : qui est un environnement multifenêtres en réseau avec des applications commandées par une souris.

Le travail en ligne de commande (mode texte) est souvent beaucoup plus efficace, plus rapide et plus approprié qu'à travers une interface graphique (dans de nombreux contextes (serveurs, systèmes embarqués, liaisons distantes lentes) on ne dispose pas d'interface graphique;).

5.1. Syntaxe des lignes de commande Unix

Les lignes de commande **Unix** peuvent être simples et se réduisent à un seul terme (ex : commande date), comme elles peuvent être plus complexes et nécessitent plus d'informations que le simple nom de la commande.

Une commande Unix peut avoir ou non des arguments. Cet argument est soit une option soit un nom de fichier .Une ligne de commande Unix s'écrit généralement sous le format suivant :

Commande *option(s) fichier(s)*

Exemple : man man

- Les ***options*** modifient, ou précisent, le comportement de la commande. Elles sont souvent de simples lettres précédées de tiret (-) et séparées par des espaces ou des tabulations. Les options des commandes peuvent souvent être regroupées :

ls **-r -l** ⟺ ls **-rl**

ou permutées :

ls **-rl** ⟺ ls **-lr**

- L'argument ***fichier*** est le nom d'un fichier que nous désirons utiliser

En général :

- Les noms de commandes sont entrés en caractères minuscules.
- Les commandes, les options et les noms des fichiers sont séparés par des espaces.
- Les options sont placées avant les noms des fichiers (la majorités des systèmes Unix permettent de traiter plusieurs fichiers).

5.2. Commandes de manipulation des fichiers

Tableau 2 : Commandes de manipulation de fichiers

Commande	Description	Options	Paramètres
pwd	affiche la référence absolue du répertoire courant		
cd	Change le répertoire courant		chemin d'accès à un répertoire
basename	extrait le nom du fichier d'un chemin complet		Chemin du fichier +[suffixe] : partie à supprimer de la chaîne

chmod	change les permissions en lecture, écriture, exécution **chmod** [ugoa] {+-} [rwx] <**fichier(s)**>	-**R** change les droits récursivement à partir du noeud	Mode (ugoa) - nom de fichier ou de répertoire
chgrp	change le groupe propriétaire du fichier. **chgrp** *option groupe* <**fichier(s)** >		Groupe – nom du fichier
chown	change l'utilisateur propriétaire du fichier **Chown** *utilisateur* < **fichier(s)**>		Utilisateur –nom du fichier
cp	copie du fichier source vers la destination	-**i** : demande confirmation -**p** : conservation de date et droits -**r** : recopie récursive d'un répertoire	source - destination
ls	Affiche la liste des fichiers dans le dossier courant ou d'un autre dossier.	-**a** : prise en compte des fichiers cachés -**F** : renseigne sur le type de fichier (*, /, @) -**i** : précision du numéro d'inode des fichiers -**R** : liste récursivement le contenu du répertoire -**l** : informations détaillées -**g** : ajout du nom du groupe -**d** : renseigne sur le répertoire lui-même -**t** : liste par date de modification -**u** : liste par date d'accès -**r** : ordre inverse	nom de répertoire
mkdir	crée des répertoires		nom du répertoire
rm	détruit des fichiers ou des répertoires	-**f** : force la commande sans s'occuper des droits -**i** : demande confirmation -**r** : destruction récursive	Nom du fichier ou du répertoire
rmdir	détruit un ou des répertoires vides		nom de répertoire

mv	déplace (ou renomme) des fichiers	-**i** : demande confirmation	Source-destination
touch	met à jour les dates d'accès des fichiers Crée fichier vide s'il n'existe pas	-**a** : dernier accès seulement -**m** : dernière modification seulement	
df	affiche la place disque disponible	**-k :** donne l'espace disque utilisé et libre sur les différentes partitions.	
du	donne la place disque utilisée par le répertoire	**-s** *:* affiche seulement l'espace total utilisé pour chaque argument en ajoutant un suffixe correspondant à l'unité (K, M, G).	Répertoire
file	donne le type de fichier		Fichier

5.3. Commandes de filtre en mode texte

Tableau 3 : Commandes de Trie

Commande	Description	Options	Paramètres
cat	Affiche le texte d'un fichier ou concatène plusieurs fichiers sur la sortie standard	**-** `v`*:* permet de convertir les caractères spéciaux des fichiers binaires en caractères affichables **- n** : numérote les lignes **- b** : numérote que les lignes non vides **- E** : affiche le symbole `$` (dollard) à la fin de chaque ligne **- s** : n'affiche au plus un ligne vide **- T** : affiche les caractères tabulations comme `^I` **- A** : équivalent à -vET **- e :** équivalent à -vE **- t :** équivalent à -vT.	Fichier à lire+fichiers
sort	trie les lignes de texte en entrée	**-** `b` : ignore les espaces en début de	fichier

		ligne - **d** : ordre alphabétique (A-Z, a-z, 0-9, *espace*) (par défaut) - **f** : ignore la casse - **n** : ordre numérique - **r** : inverse l'ordre de tri.	
sed	effectue des modifications sur les lignes de texte sed [-n] [-e commande] [-f fichier de commandes] [fichier]	- **n**: écrit seulement les lignes spécifiées (par l'option /p) sur la sortie standard. **-e**:permet de spécifier les commandes à appliquer sur le fichier. Cette option est utile lorque vous appliquez plusieurs commandes. Afin d'éviter que le shell interprète certains caractères, il faut mieux encadrer la commande avec des ' ou des " ; - **f**:les commandes sont lue à partir d'un fichier	
more	affiche à l'écran l'entrée standard en mode page par page		
less	affiche à l'écran l'entrée standard en mode page par page avec des possibilités de retour en arrière		
cut	permet d'isoler des colonnes dans un fichier		
expand	transforme les tabulations en espaces		
head	affiche les n premières lignes		
join	permet de joindre les lignes de deux fichiers en fonction d'un champ commun		

paste	concatène les lignes des fichiers		
tail	affiche les n dernières lignes du fichier		
tac	concatène les fichiers en inversant l'ordre des lignes	*	
uniq	élimine les doublons d'un fichier trié		
rev	inverse l'ordre des lignes d'un fichier		
pr	formate un fichier pour l'impression		
tee	écrit l'entrée standard sur la sortie standard et dans un fichier		
tr	remplace ou efface des caractères		

5.4. Gestion des processus

Tableau 4 : Commandes de gestion de processus

Commande	Description
ps	affiche la liste des processus
kill	envoie un signal à un processus
nice	fixe la priorité d'un processus

5.5. Commandes diverses

Tableau 5 : Autres commandes

Commande	Description
true	ne fait rien sans erreur

false	ne fait rien, avec une erreur
echo	affiche une ligne de texte
date	affiche et modifie la date courante
gzip	compresse les fichiers
sleep	attend pendant le temps spécifié
grep	recherche de chaînes de caractères dans des fichiers
find	recherche des fichiers sur le système
bc	calculatrice en mode texte
cal	affiche le calendrier
clear	efface l'écran
csplit	découpe un fichier en sections
diff	permet d'obtenir la différence entre deux fichiers
factor	recherche les facteurs premiers d'un nombre
hexdump	affiche le contenu d'un fichier en hexadécimal
id	affiche l'identification d'un utilisateur
passwd	permet de changer le mot de passe
wc	compte les caractères, les mots et les lignes en entrée
whereis	permet de trouver l'emplacement d'une commande
who	affiche la liste des utilisateurs présents

Chapitre 2 : La programmation Shell

1. Pourquoi programmer en shell ?

Un interpréteur est un programme interactif servant d'intermédiaire entre l'utilisateur et les programmes binaires exécutables. Cet interpréteur qui appelle le système quand il le faut, entoure l'ensemble des commandes .Il s'appelle le shell. Il interprète les commandes que lui fait l'utilisateur. Comme ces demandes se traduisent souvent par un appel à une commande, nous disons qu'il interprète un langage de commande. Par la suite et par abus de langage les termes interpréteurs du langage de commande et langage de commande pourront être confondus.

Ainsi ; nous pouvons dire que le shell est le nom générique donné aux langages de commandes (en mode interactif ou en mode script) .En effet, shell ne se limite pas à un simple interpréteur de commandes, mais dispose d'un véritable langage de programmation qui permet d'écrire des programmes d'enchaînement de commandes : Scripts.

Ces scripts permettent à l'administrateur de simplifier la réalisation de tâches répétitives. Un script est un fichier texte contenant les commandes à exécuter ainsi que, éventuellement, des structures de contrôle, des boucles, des assignations de variables, des fonctions et des commentaires.

De nombreux shells sont accessibles sous Linux, mais c'est généralement le bash (Bourne Again SHell) qui est utilisé pour la programmation shell car il est facile d'emploi, libre de droit et gratuit.

Les divers langages existants sont très semblables : bash, csh, ksh ou sh (utilisé ici). Le même mot désigne l'interpréteur de commandes (shell) et le langage dans lequel les commandes sont écrites (Shell).

1.1. Niveau lexical :

Le shell interprète des instructions du langage de commandes qui doivent être séparées par des séparateurs d'instruction. Comme il y a plusieurs sortes d'instructions, nous utiliserons le vocabulaire suivant :

- Une commande est une instruction analogue à un appel de sous-programme externe, dont le code est exécutable par la machine (cp ,mkdir et rm sont des commandes).

- Une commande interne est une instruction analogue à un appel de sous-programme interne au shell, son action est réalisée par le shell lui-même (exit, read sont des commandes internes du Shell).
- Une instruction est une commande interne qui est une structure de contrôle (if ,while sont des instructions du Shell)
- Une phrase ou une ligne de commande est une composition syntaxiquement correcte de commandes, commandes internes et instructions, que sait interpréter le Shell

1.2. Les séparateurs :

Les séparateurs de phrases (lignes de commande) sont le RET ou le point-virgule. Dans une même phrase, les séparateurs d'entités lexicales sont le blanc et le caractère de tabulation(TAB).Le nombre de séparateurs entre entités est quelconque.

1.3. Les Mots :

Les mots du langage sont constitués avec les caractères lettres, chiffres et souligné, ils commencent par une lettre ou un souligné. Les mots désignent les variables, les symboles du langage (mots-clé) et les noms et les noms des fichiers.

1.4. Les Caractères génériques:

Les caractères génériques, nommés également des jokers, sont des caractères spéciaux qui ont la capacité d'être remplacés par des chaines en fonction des noms des fichiers du répertoire courant. Ces caractères sont l'étoile, le point d'interrogation et le crochet ouvrant associé au crochet fermant.

- ? Remplace un caractère quelconque, sauf le RET ;
- Remplace n'importe quelle chaine, même vide, de caractères ne comportant pas de RET.
- [ ] Remplace un caractère parmi ceux qui sont énumérés entre les crochets, ou un caractère parmi ceux qui ne sont pas énumérés entre les crochets si le premier d'entre eux est un point d'exclamation.

1.5. Les Caractères spéciaux :

Le caractère de continuation \ placé en fin de ligne indique que la phrase en cours n'est pas terminée et se poursuit sur la ligne suivante. Le RET est alors transformé en espace.

Les caractères spéciaux ont une signification dans le langage, à moins qu'ils ne soient ignorés. Un caractère spécial devient un caractère normal s'il est précédé du caractère \ ou s'il est situé dans un parenthésage particulier. Les caractères spéciaux sont :

& | <>$ ( )’ " ‘{} \

Ainsi, un \ en fin de ligne annule le retour-chariot, et c’est pourquoi l’antislash est considéré comme le caractère de continuation d’une ligne.

1.5.1. Les commandes bloquantes et non bloquantes : Opérateur &

Unix est un système d'exploitation multi-tâche, ce qui signifie que vous pouvez lancer plusieurs commandes simultanément. Unix dispose d'un gestionnaire de tâches dont le rôle est de contrôler l'exécution simultanée des programmes.

Chaque programme est appelé **processus** ou ***tâche***. Par exemple éditer ou imprimer un fichier sont des tâches (ou processus). Une tâche peut comporter plusieurs programmes connectés par des tubes. Plusieurs commandes sont disponibles pour gérer l'exécution des tâches.

- Exécuter une tâche au ***premier plan (foreground)*** signifie que pendant son exécution l'utilisateur n'a pas la main.
- Une exécution en ***arrière plan (background)*** signifie que la tâche est exécutée de façon parallèle et la main est rendue à l'utilisateur qui peut lancer d'autres commandes.

Pour exécuter une commande en arrière-plan, il suffit d'ajouter le signe & à la fin de la ligne avant de taper la touche Return.

Exemple

cp F G → processus en *foreground*, commande bloquante
cp F G & → processus en *background*, commande non bloquante

1.5.2. Les redirections d'entrée / sortie

Quand les commandes sont interprétées :

- Elles affichent leurs résultats sur ***l'unité de sortie standard*** qui est par défaut l'***écran***.
- Elles reçoivent leurs informations de ***l'unité d'entrée standard*** qui est classiquement ***le clavier***.
- Les messages d'erreurs sont envoyés sur l***a sortie d'erreur***. Par défaut, la sortie d'erreur est renvoyée à l'écran.

Il est possible de contrôler les entrées et sorties standard et ainsi de redéfinir la destination des informations fournies par les commandes. De cette manière, ces informations peuvent être placées dans des fichiers et utilisées ultérieurement.

A. Rediriger la sortie vers un fichier : Opérateur >

Le symbole > permet d'envoyer le résultat de la commande dans un fichier. Dans l'exemple suivant le résultat de la commande ***ls -la*** est renvoyé dans le fichier *liste-fichiers* :

ls -la > liste-fichiers

Lorsqu'une sortie est redirigée vers un fichier, les informations ne sont plus affichées à l'écran. Dans le cas d'une redirection vers un fichier, si le fichier existe il est ***écrasé***, s'il n'existe pas il est automatiquement créé.

B. Rallongement d'un fichier : Opérateur >>

Les deux caractères >> indiquent que les informations doivent être ajoutées à la fin d'un fichier existant.

La commande **cat clients >> nouvclients** ajoute le contenu du fichier *clients* à la fin du fichier *nouvclients*. Si le fichier *nouvclients* n'existe pas il est créé.

C. Rediriger l'entrée d'une commande : Opérateur <

Il est également possible de prendre l'entrée d'une commande dans un fichier en utilisant le symbole < . Dans l'exemple suivant, la commande ***mail gdurand*** prend ses informations dans le fichier *info*.

mail gdurand < info

Ainsi l'utilisateur *gdurand* reçoit un courrier dont le texte est le contenu du fichier *info*.

D. Rediriger les messages d'erreurs : Opérateur 2>

Unix utilise un canal spécial pour afficher les messages d'erreurs. Pour envoyer ces messages d'erreurs dans un fichier, on utilise la syntaxe **2>**.

L'exemple suivant renvoie les éventuelles erreurs de la commande ***cat*** dans le fichier **erreurs** : cat fichier-test **2>** erreurs

Notez que les résultats de la commande s'affichent toujours à l'écran (sortie standard). Le tableau suivant donne la liste des différentes syntaxes de redirection.

Tableau 6 : Syntaxe de redirection des entrées /Sorties sous unix

Syntaxe	Fonction	Exemple
>	Redirection de la sortie standard	cat fichier1>fichier2
>>	Ajoute les sorties à un fichier existant	cat fichier1>>fichier2
2>	Redirige les messages d'erreur vers un fichier	cat fichier1 2> erreurs
2>>	Ajoute les messages d'erreur à un fichier existant	cat fichier1 2>> fichier2
<	Redirection des entrées	tr aA < fichier2

1.5.3. Notion de Pipe : Opérateur |

Le pipe (ou tube) redirige la sortie (résultat) d'un processus directement à l'entrée d'un autre.

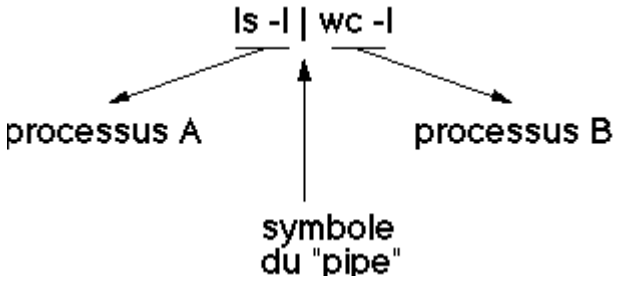

Figure 4 : Exemple d'utilisation de pipe(tube)

Dans la Figure 4 Le résultat de la commande de gauche est redirigé en entrée de la commande de droite. Logiquement cela équivalent à :

- ls -l > F
- wc -l < F

- En utilisant le pipe, nous ne créons pas de fichier F : ls et wc sont deux processus s'exécutant en pseudo-parallélisme. La synchronisation est réalisée par le système, en suspendant **wc** quand le pipe est vide, et en suspendant **ls** quand le pipe est plein.

Remarque : les pipes sont créés par le système et sont unidirectionnels.

Exemple :

1- ls | wc : permet de calculer le nombre de fichiers du répertoire courant.
2- ls | grep toto : permet de chercher les fichiers du répertoire courant qui contiennent la chaîne de caractère *toto*.
3- who | wc - l : permet de calculer le nombre de personnes connectées sur la machine.

1.5.4. Parenthésages:

Les parenthésages servent à grouper des séquences de mots ou à isoler des mots :

- {…} regroupent de commandes, parenthésage ordinaire ; délimite aussi la porté de l'opérateur de valeur $.
- (…) entoure une liste de commandes qui sont exécutés par un nouveau processus UNIX.
- '… ' (les quotes) c'est un parenthésage dans lequel aucun caractère sauf ' n'a de signification spéciale.
- "…" (Guillemets) parenthésage dans lequel aucun caractère sauf ',' et $ n'a de signification spéciale ;

- ‘…‘ (accents graves) exécution de la phrase placée entre accents graves, délivre la sortie standard des commandes situées dans ce parenthésage. Les caractères spéciaux à l’intérieur de parenthésage ne soient interprétés qu’à l’exécution de la phrase. Ce parenthésage correspond un peu à une activation de commandes en tan qu’appel de fonction, puisqu’il délivre une valeur qui lui est substituée dans le texte.

2. Le premier Programme :

Pour écrire nos programmes shell nous utilisons un éditeur de texte, tel que nedit, kedit, emacs, vi...

```
#!/bin/sh
# fichier : PremierProgramme.sh
# Affiche un salut a l'utilisateur
echo "Bonjour $USER"
```

Chaque programme doit commencer par la ligne suivante (c'est obligatoirement la première ligne du fichier) : *#!/bin/sh.* elle permet de fournir au noyau la commande à lancer pour exécuter le script.

Elle permet de dire que l'argument suivant est le programme à utiliser pour exécuter ce fichier. Ici, c’est le shell /bin/sh. Lorsque notre script est terminé et enregistré, nous devons le rendre exécutable pour pouvoir l'utiliser à l'aide de la commande *chmod*. On tape alors :

chmod +x PremierProgramme.sh

Nous pouvons maintenant lancer notre script en tapant : ./PremierProgramme.sh

3. Les commentaires

Les commentaires, en langage shell, commencent par le caractère # et vont jusqu'à la fin de la ligne.

4. Les variables

4.1. Les variables en Shell :

Comme dans tout langage de programmation, nous ne pouvons pas nous surpasser des variables.

En langage shell, les variables sont désignées par les mots ;elle ne se déclarent pas , et leur type (numérique ou chaine) dépend de leur interprétation par les commandes. Dans un même processus, toutes les variables doivent avoir des noms différents.

Pour initialiser une variables et lui affecter une valeur, il suffit d'écrire :

Nom_variable=valeur

Il ne faut pas qu'il y ait espaces de chaque coté du symbole=.Une valeur peut être une valeur numérique, une chaine, le résultat d'une commande…...

Pour récupérer la valeur de la variable, il suffit d'écrire le signe dollar $ devant le nom de la variable : $

```
#!/bin/sh
# affecter une valeur à la variable (sans la déclarer) :
a="Bonjour $USER "
# afficher le contenu de la variable "a" :
echo "La valeur de a est :"
echo $a
```

Pour ne pas avoir de confusion entre les variables et le reste du texte ; il faut utiliser les accolades :

```
#!/bin/sh
# affecter une valeur à la variable (sans la déclarer) :
a="Bonjour $USER "
# afficher le contenu de la variable "a" :
echo "La valeur de a est : {$a}Bienvenu"
```

Si nous écrivons le programme précédant sans utiliser les {} ; c-à-d : *echo "La valeur de a est : $aBienvenu"* .Le programme donnera comme résultat :< La valeur de

a est :> car le shell recherchera une variable nommée *aBienvenu* qui n'a pas de valeur définie.

Il existe quelques variables définies par le système et utilisées par le shell lui-même ou par les processus en exécution. On cite par exemple:

- le chemin du répertoire de base de l'utilisateur : $HOME
- Une suite de chemins d'accès aux répertoires des exécutables: $PATH
- l'invite de l'interpréteur : $PS1
- le nom de l'utilisateur : $USER
- le type de shell : $SHELL
- le nom du type du terminal : $TERM
- liste de chemins d'accès pour la commande cd : $CDPATH
- chemin d'accès à la boîte aux lettres utilisateur : $MAIL
- intervalle en sec au bout duquel le mail est : $ MAILCHECK

La commande set (sans paramètre) permet de visualiser les variables et leurs valeurs. Une commande peut être effacée de l'environnement par la commande Unset (ex unset variable).

La commande export permet de définir des variables d'environnement qui pourront être utilisées par plusieurs processus fonctionnant simultanément. Cette commande permettra de faire connaître alors la variable aux processus fils du shell. Les variables créées par le fils ne sont pas transmises au père.

Il existe des méthodes d'accès plus sophistiquées permettant une substitution de valeur. La substitution de valeur permet d'évaluer conditionnellement la valeur du variable. Elle sert à donner des valeurs par défaut à des variables dont on ne sait pas si elles sont initialisées. Tableau7 montre ce mécanisme :

Tableau 7 : La substitution de valeurs sous UNIX

Méthode	Résultat
${variable:- valeur}	donne la valeur de la variable si celle-ci n'est pas nulle, sinon valeur de substitution
${variable:= valeur}	comme ci-dessus et la valeur de substitution est affectée à la vari
${variable:?message}	donne la valeur de la variable si celle-ci n'est pas nulle, sinon message de substitution et termine le processus shell en cours
${variable:+valeur}	donne la valeur de substitution si la variable n'est pas nulle, donne rien

4.2. Les variables prédéfinis :

Il existe plusieurs variables prédéfinies utilisables dans les scripts :

Tableau 8 : Variables prédéfinis

Variable	Fonction
$$	l'identificateur du processus courant (pid)
$?	la valeur de sortie de la commande précédente(code de retour d'une com
$!	l'identificateur du processus fils (numéro du dernier processus lancé e plan)
$0	donne le nom de fichier du script courant
$1 à $9	les neuf premiers arguments du script
$#	le nombre d'arguments d'une commande
$*	tous les arguments à partir de $1 séparés par un espace

Les variables $1 à $9 contiennent les neuf premiers arguments du script. Pour accéder aux autres, il faut utiliser la commande shift. Celle-ci décale les arguments d'un cran vers la gauche : $2 est mis dans $1, $3 dans $2, etc. La variable $9 est initialisée avec le dixième argument.

5. Les opérations

La commande *expr* permet d'effectuer des opérations arithmétiques entières sur les variables.
ATTENTION : il faut toujours banaliser les méta caractères (caractères génériques ou spécifiques) utilisés dans la commande *expr* avec \ .

```
$ expr 4 + 3
7
$ a=5
$ expr 4$a + 1
```

```
46
$ expr $a \< 7
1
$ expr a \< c
1
```

Les opérations possibles sont :

- + - * / % (reste) sur des opérandes entiers ;
- < > = >= <= ! (différent de) sur des opérandes quelconques, avec comparaisons faites suivant l'ordre lexicographique entre chaînes de caractères ;
- | & opérations booléennes.

La priorité est la priorité usuelle. On peut grouper avec des parenthèses, qu'il faut penser à banaliser.

```
$ a=`expr 2 \* \(  3 + 5  \)` (affectation)
$ echo $a
16
$ a=`expr $a + 1` (incrémentation)
$ echo $a
17
```

Attention : pour l'affectation la commande *expr* doit être mise dans le parenthesage `..`

6. Les tests

Il y 'a deux façons de faire des tests : Utiliser la commande test *expression* ou bien [*expression*](Vous devez impérativement avoir un espace autour de chaque crochet)

Cette commande renvoie 0 si expression est vraie et un autre entier sinon.

6.1. Tests sur les chaînes de caractères :

- test -z chaîne est vrai ssi chaîne est vide
- test -n chaîne est vrai ssi chaîne est non vide
- test chaîne1 = chaîne2
- test chaîne1 != chaîne2

6.2. Test des nombres

Si deux chaînes n et m contiennent des entiers sous forme décimale, la commande test permet d'effectuer les comparaisons suivantes :

- [n -eq m] est vrai ssi n est égal m.
- [n -ne m] est vrai ssi n est différent de m.
- [n -ge m] est vrai ssi n est plus grand ou égal à m.
- [n -gt m] est vrai ssi n est strictement plus grand que m.
- [n -le m] est vrai ssi n est plus petit ou égal à m.
- [n -lt m] est vrai ssi n est strictement plus petit que m.

6.3. Tests sur les fichiers :

- test –f fichier est vrai ssi fichier est un fichier
- test -d fichier est vrai ssi fichier est un répertoire
- test -r fichier est vrai ssi fichier est autorisé en lecture
- test -w fichier est vrai ssi fichier est autorisé en écriture
- test -x fichier est vrai ssi fichier est autorisé en exécution
- test -s fichier est vrai ssi fichier est de taille non nulle

Les tests sont rarement utilisés seuls (bien que cela soit possible).Ils le sont en général dans une conditionnelle.

7. L'exécution conditionnelle

7.1. l’insruction if hen fi

L'exécution conditionnelle est gérée par la structure :

```
if commande
then
    autres commandes
fi
```

L’instruction if teste le code de sortie de la commande en paramètre. Si celui-ci vaut 0 (vrai), le script exécute les commandes entre then et fi.

Il est possible d'avoir des structures du type :

```
if commande
then
        commandes
else
        commandes
fi
```

ou :

```
if commande
then
        commandes
elif commande
        commandes
fi
```

Attention : les mots-clés then, else et fi doivent être en début de ligne ou précédés de ;

Exemple1

```
#!/bin/sh
if [ "$SHELL" = "/bin/bash" ]
 then
  echo "Vous travaillez avec le shell : bash "
else
  echo " Vous travaillez avec le shell : $SHELL"
fi
```

Exemple2

```
#!/bin/sh
if [ -f $1 ];then if [ -x $1 ]
then echo $1 est un exécutable
```

```
else echo $1 n est pas un exécutable
fi
else echo $1 n‘est pas un fichier
fi
```

Il existe deux commandes utiles pour les tests, ce sont true qui retourne toujours 0, donc vrai, et false qui retourne toujours 1, donc faux.

7.1.1 combinaison de plusieurs conditions

Si nous voulons faire le test sur plusieurs conditions , nous devons les combiner de la façon suivant :

- [cond1 -a cond2] : Vrai seulement si les deux conditions sont remplies.
- [cond1 -o cond2] : Vrai dès que l'une des deux conditions est remplie.

Exemple :

```
#!/bin/sh
a=$1
b=$2
if [ \( $a != $b \) -a \( $a -gt $b \) ]
then echo $a supérieur a $b
else echo $a plus petit que $b
fi
```

7.1.2. Les Opérateurs Condensés (Shortcut Operators)

Pour permettre des tests complexes mettant en œuvre plusieurs commandes, il faut utiliser les opérateurs && et || qui effectuent un ET logique et un OU logique sur le résultat des commandes. Ces opérateurs sont utilisés de la façon suivante :

```
commande1 && commande2
commande1 || commande2
```

Dans le cas du ET logique, la deuxième commande ne sera exécutée que si la première réussit. Dans le cas du OU logique, la deuxième commande ne sera exécutée que si la première échoue.
La syntaxe && peut être utilisée comme un raccourci de l'instruction if. La partie droite de l'instruction est exécutée si la partie gauche est vraie

Exemple : (Or)

```
#!/bin/sh
mailfolder=/var/spool/mail/toto
[ -r "$mailfolder" ] || { echo "Je ne peux pas lire $mailfolder" ; exit 1; }
echo "$mailfolder a recu un courrier de :"
grep "From " $mailfolder
```

Ce script teste tout d'abord s'il peut lire le dossier du courrier de l'utilisateur toto précis. Si c'est le cas, il affiche les lignes "From" du dossier. S'il ne peut pas lire le fichier $mailfolder, alors l'opérateur "OR" entre en action.Le problème, ici, c'est qu'après OR nous ne pouvons mettre qu'une seule commande .Ici , nous avons utiliser le parenthésage {…} pour transformer, les deux commande **echo** et exit en une seule commande.

Vous pouvez tout faire sans les opérateurs AND et OR, uniquement en utilisant les instructions if, mais souvent les raccourcis des opérateurs ci-dessus sont plus pratiques.

Exemple : (ET)

```
#!/bin/sh
[ -f exemple ] && echo "exemple est un fichier "
```

Nous pouvons lire l'exemple comme suit: "exemple est un fichier ET la commande echo est exécutée"

7.2. L'instruction Case

L'autre commande d'exécution conditionnelle est la commande case qui permet de comparer une valeur à plusieurs expressions et d'exécuter les commandes lorsqu'il y a égalité. La syntaxe de cette commande est :

```
case valeur in
    expression1) commande
                    autre commande ;;
    expression2) commande
                    autre commande ;;
   *) commande
      aure commande ;;
    esac
```

Si une expression correspond à la valeur, les commandes sont exécutées et $? est mis à 0. Sinon $? est mis à 1.

Par exemple, le code suivant teste l'argument donné en paramètre :

```
case $1 in
  oui | o | OUI | O ) echo "OK" ;;
  non | n | NON | N ) echo "Pas d'accord" ;;
  * ) echo "OUI ou NON" ;;
esac
```

8. L'exécution itérative

8.1. L'instruction While

L'interpréteur de commandes gère trois types de constructions pour les boucles :

```
while commande
do
commandes
done
ou
until commande
do
commandes
done
```

ou

```
for variable in liste_de_valeurs
do
commandes
done
```

Pour les deux premières constructions, le fonctionnement est similaire à la commande if. Pour la boucle for, la variable prendra successivement toutes les valeurs de la liste.

Les commandes break et continue permettent de modifier le fonctionnement des boucles. La commande break est utilisée sous deux formes :

- break : sort de la boucle courante
- break n : n étant un entier supérieur à 0, permet de sortir de n boucles imbriquées.

La commande continue permet de passer à l'itération suivante sans exécuter les commandes situées après.

9. Re-direction et structures de contrôle

Il est possible de rediriger la sortie ou l'entrée d'une structure de contrôle globalement. Par exemple, le programme suivant lit les lignes à partir d'un fichier dont le nom est dans la variable $fichier et imprime celles dont la longueur est supérieure à deux caractères dans le fichier temporaire tmp.txt.

```
while read ligne
do
        longueur=`echo $ligne | wc -c`
        if [ $longueur -lt 2 ]
        then
                continue
        fi
        echo $ligne
done < $fichier > tmp.txt
```

10. Les fonctions

Le shell permet de définir des fonctions appelables par le script. Cette méthode est plus efficace que d'appeler des scripts externes car la fonction est exécutée dans le même processus. La syntaxe de définition d'une fonction est :

```
fonction()
{
        commandes
}
```

et elle est appelée par :

```
fonction [paramètres]
```

Les fonctions reçoivent les paramètres d'appel dans les variables $1 à $9 et $*. La valeur de $0 reste inchangée.

Il est possible de déclarer des variables locales à une fonction en utilisant la commande local avant la déclaration de la variable. Dans ce cas, la modification de la variable locale ne modifie pas une variable globale portant le même nom.

Exemple :

```
MaFonction()
{
        local maVariable="coucou"
        echo $maVariable
}
maVariable="Bonjour"
echo $maVariable
MaFonction
echo $maVariable
```

La commande return permet de spécifier le code de retour de la fonction. S'il n'est pas spécifié, la variable $? est initialisée au code de fin de la dernière commande de la fonction.

Le shell supporte les appels récursifs de fonctions. Comme dans tout langage récursif, il faut faire très attention aux effets de bord et aux variables non déclarées en local.

TP1-Prise en main d'Unix

Le but de ce premier TP est de commencer à vous familiariser avec l'environnement Unix.

1- Introduction

Pour interagir avec un système UNIX, l'utilisateur rentre des commandes dans un interpréteur : le shell. Vous pouvez lancer cet interpréteur à l'aide d'un programme nommé console ou terminal (xterm). Le premier caractère de début d'une ligne dans cet interpréteur est le prompt (généralement c'est le caractère $, qui peut etre précédé du nom de l'utilisateur ainsi que d'un chemin dans le système de fichier, par exemple :

nom@machine:~$

⟶ L'interpréteur est prêt à recevoir une commande. Taper cette commande au clavier puis appuyer sur la touche <RET> (retour chariot, ou touche "entrée") permet d'exécuter une commande.

Les commandes UNIX suivent le format suivant : Commande *option(s) fichier(s)*

nom@machine:~$ Commande *option(s) fichier(s)*

- *Commande* : le nom de la commande à exécuter,
- *options* : est une liste d'options (souvent une suite de lettres ou des mots qui suivent séparés par un tiret " - ") pour cette commande,
- *fichier(s)* : est l'entité ou les entités sur lesquelles la commande sera exécutée.

Remarque : UNIX respecte la casse,

Avant de commencez par vous familiariser avec les différents menus de votre espace de travail Unix : Il faut d'abords se connecter à cet espace.

2- Connexion au système

Chaque utilisateur est identifié par une procédure de login : nom d'utilisateur (***username***) valide suivi du mot de passe (***password***) correspondant. C'est l'administrateur système qui assigne le nom d'utilisateur et le mot de passe initial. Chaque station a un nom de machine (***hostname***).

Nous utilisons pour nos Tps des machines sous Linux (**ubuntu**).

Se connecter est évidemment la première chose à faire. Le système démarre et affiche une fenêtre vous invitant à taper votre identifiant (**login**) et votre mot de passe (**password**), ce qui vous permet d'ouvrir une session.

Dans les salles machines de la faculté et par raison de sécurité, l'étudiant accède au système **ubuntu** comme utilisateur invité et donc il ne passe pas par la procédure

d'authentification. Autant qu'invité, l'étudiant n'aura pas tous les droits d'accès mais pourra exécuter les commandes de base.

3 - Découverte de l'interface graphique

Une fois votre nom d'utilisateur et mot de passe vérifiés (sur une version installée d'unix), le gestionnaire de bureau, le programme qui gère l'affichage des menus et des fenêtres (ici KDE) apparaît. Le menu principal (bouton K en bas à gauche) permet de lancer des applications, d'accéder aux outils de paramétrage du système ou encore de fermer la session ou d'éteindre l'ordinateur.

4- Ajouter un utilisateur

Note : la plupart des programmes ainsi que le menu principal disposent d'une entrée « aide » (help) permettant d'accéder à l'aide en ligne. Si vous êtes coincé(e), n'hésitez pas à la consulter.

Explorez un peu le menu principal. Lancez quelques applications, et exercez-vous à agrandir, fermer et déplacer les fenêtres. Parcourez l'ensemble des menus disponibles et essayez de deviner à quoi sert chaque fonction.

Avec le bouton droit de la souris, vous pouvez faire apparaître différentes commandes selon l'endroit cliqué (on parle de menus « contextuels »).

Si vous voulez ajouter un utilisateur il faut basculer vers le ***root*** (super utilisateur) car il est le seul utilisateur qui a le droit d'ajouter ou de modifier les paramètres d'un utilisateur.

Pour passer au mode ***root*** ouvrez une console et tapez ***su*** ou ***sudo***. Tapez ensuite ***adduser*** pour ajouter un nouvel utilisateur.

Par défaut, les users invités n'ont pas de ***password***. Pour mettre un mot de passe root tapez : ***sudo passwd*** et pour mettre un mot de passe au user "knoppix", tapez: ***sudo passwd knoppix***.

Vous pouvez visualiser les utilisateurs que vous avez ajouté en ouvrant le fichier ***/etc/passwd***.

5- Découvrez l'arborescence des fichiers Unix

Ouvrez la racine / et découvrez l'arborescence d'Unix en parcourant les différents répertoires qui s'y trouvent et en se rappelant (cours) le types de fichiers ou de répertoires qu'ils contiennent.

6- Ajout fichiers/répertoires

- Créer un utilisateur portant votre nom.
- créer une arborescence de fichiers et répertoires dans votre répertoire home(en utilisant la souris).

TP2- Commandes de base d'Unix

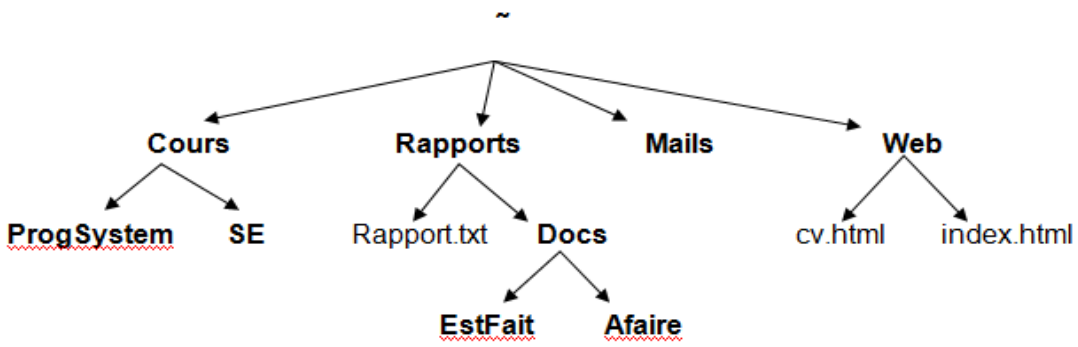

Figure 1 : Arborescence des fichiers

Parite I : Manipulation des fichiers et des répertoires

1. Dans quel répertoire se trouve l'utilisateur à l'ouverture de sa session de travail ?
2. Créer l'arborescence de la *Figure1*. Le ˜ représente le répertoire personnel de l'utilisateur. Les répertoires apparaissent en gras.
3. Utilisez la commande ***mkdir*** pour créer des répertoires vides : Utiliser la commande ***touch*** pour créer un fichier vide ou bien ***cat*** en utilisant la redirection **d'E/S(>).**
4. Aller directement dans *˜/Rapport/Docs/Afaire/.*
5. De là, passer dans *˜/Rapport/docs/EstFait*/.
6. Revenir dans *˜/Rapport*/.
7. Sans changer de répertoire, regarder avec ***less*** le contenu de *˜/Web/index.html*.
8. Passer dans *˜/Rapport/Docs/*.
9. En restant au même endroit, afficher le contenu du répertoire *Mail*.
10. Revenir dans le répertoire principal.
11. Ouvrez ˜/*Rapport*/.
12. Affichez le contenu du fichier Rapport.txt en utilisant la commande ***cat***.
13. Ecrivez la phrase suivante dans le fichier *Rapport.txt* en utilisant la commande ***echo*** : " *C'est ma première phrase dans ce fichier* "
14. Réaffichez le contenu du fichier *Rapport.txt*.

15. Réécrivez la phrase précédente dans le fichier et réaffichez le contenu du fichier. Quesque vous remarquez ?
16. Insérez la phrase suivante dans le même fichier sans effacer la première : " *C'est ma deuxième phrase dans ce fichier* ".
17. Afficher le chemin absolu de votre répertoire courant.
18. Créer un fichier *Fich* dans le répertoire *AFaire* contenant le texte suivant :

aaaaaaaaa
bbbbbbbbbbbbb
ccccc
ddddddddddddd
eeeeeeeeeeee
fffffffffffffffffff
ggggggggggggg
hhhhhhhhhhhhhhhhhh
iiiiiiiiiiiiiiii
jjjjjjjjjjjjjjjjjjjjjjjjjjjjj
kkkkkkkkkkkkkkkkkkk

a. Affichez les dix premières lignes du fichier *Fich* avec la commande ***head***.
b. Affichez les deux premières lignes du fichier *Fich.*
c. Affichez les 10 dernières lignes du fichier *Fich* avec ***tail***
d. Affichez les 3 dernières lignes
e. Affichez de la cinquième à la dernière ligne
f. Affichez les lignes du fichier numérotées avec la commande ***nl***.
g. Affichez le nombre de lignes, mots et caractères du fichier *Fich* en utilisant ***wc.***
h. Affichez juste le nombre de lignes puis juste le nombre de caractères et enfin affichez le nombre de caractères et de mots du fichier *Fich*.

19. Déplacez vous dans le répertoire /*etc*
    a. Affichez le contenu du fichier *passwd* contenu dans /*etc*.
    b. Affichez les lignes de fichiers qui contiennent le mot *root* avec la commande ***grep.***
    c. Affichez le nombre de lignes contenant le mot *root*.
    d. Affichez les lignes ne contenant pas *root*.

e. Affichez les lignes ne contenant pas *root* ainsi que le numéro de la chaque ligne.

20. Passez au mode super utilisateur et créer un nouvel utilisateur portant votre nom.
    a. Afficher la ligne du fichier */etc/passwd* qui contient votre mot de login.
    b. Affichez avec la commande ***cut*** les caractères du 2ème au 5ème de toutes les lignes.
    c. Afficher les six premiers caractères et le 10ème caractère de toutes les lignes.
    d. Affichez les champs 1 et 7 du fichier *passwd* en utilisant le séparateur **:** .
21. Le répertoire */usr/include* contient les fichiers d'entête standards en langage C (stdlib.h,...).
    a. Créer un répertoire nommé *inc* dans votre répertoire de connexion. En utilisant une seule commande, y copier les fichiers du répertoire */usr/include* dont le nom commence par *std* et termine par .h avec la commande ***cp.***
    b. Afficher la liste des fichiers de */usr/include* dont le nom commence par a, b ou c et termine par **.h**.
    c. Afficher la liste des fichiers de */usr/include* dont le nom comporte 3 caractères suivis de .h. Supprimez les avec la commande ***rm***.
    d. Détruisez tous les fichiers contenus dans le répertoire *inc*.
    e. Détruisez le répertoire *inc* avec ***rmdir*** ou bien ***rm –R.***
    f. Renommez le fichier *~/Rapport/Rapport.txt* en *~/Rapport/RapportFinal.txt*.
    g. Déplacez le *~/Rapport/RapportFinal.txt* dans *~/Rapport/Docs/Afaire*. Déplacez le dans cette fois dans *~/Rapport/Docs/Estfait* en modifiant son nom en *~/Rapport/VersionFaite.txt*.
22. Créer les deux fichiers suivants *Fich1* et *Fich2* dans ~/*Rapport*/*Docs* avec respectivement les contenus suivant :

Fich 1 :	Fich 2 :
1erre ligne	1erre ligne
2eme ligne	seconde ligne
3eme ligne	3eme ligne

    a. Comparez le contenu des deux fichiers *Fich1* et *Fich2* en utilisant la commande ***cmp***.
    b. Créez un troisième fichier *Fich3* identique à *Fich1*.
    c. Constatez qu'ils sont identiques à l aide de la commande ***cmp***.
    d. Ajoutez la ligne « *4eme ligne* » dans *Fich1* et les deux lignes suivantes dans *Fich2* :

Quatrieme ligne

5eme ligne

e. Affichez les différences de contenu entre *Fich1* et *Fich2* avec la commande ***diff***.

23. Créer Le fichier *Etudiants* suivant dans *~/Rapport/Docs*.

M:Dup:Ge:38:T

F:Dura:Gl:25:A

M:Ab:Ma:42:la

M:Es:Au:17:Au

F:Durt:so:24:al

M:dura:be:33:tou

a. Triez le fichier Etudiants par ordre alphabétique en utilisant la commande ***sort***.
b. Triez le fichier à partir du deuxième champ le séparateur est :
c. Riez en ordre inverse.
d. Triez sur le 4ème champ numérique.

Parite II : Pipes (Enchaînements de commandes, une seule ligne)

24. Compter le nombre de fichiers du répertoire courant.
25. Compter le nombre de fichiers du répertoire courant.
26. Compter le nombre des répertoires du répertoire courant.
27. Afficher le plus gros fichier du répertoire courant.
28. Renvoyer le nombre d'utilisateurs connectés en utilisant la commande ***who***.
29. Compter le nombre de lignes contenant votre nom de login dans */etc/passwd*.
30. Affichez uniquement les login de tous les utilisateurs du système triés.
31. Affichez les correspondances login, numéro d'utilisateur (UID) , nom réel. Eliminez les doublons en utilisant la commande ***uniq***. (La structure du fichier */etc/passwd* est détaillée dans le chapitre 1 paragraphe 4.2.4)
32. Compter le total des lignes des fichiers dans */etc*
33. Afficher le nombre de fichiers n'appartenant pas à *root* dans */etc*.

Partie III : Droits d'accès

34. Un ensemble de permissions d'accès est associé à tout fichier. Ces permissions déterminent qui peut accéder au fichier et comment. Ainsi ; Pour chaque fichier, on peut avoir le droit de :

    r : accès en lecture

    w : accès en écriture

    x : accès en exécution

    Une combinaison arbitraire de ces droits.

    De plus, on peut donner/supprimer ces droits pour :

    u : le propriétaire du fichier,

    g : le groupe du fichier,

    o : tous les utilisateurs.

    a : tout le monde

    a. Créer un fichier vide *mon_fichier* dans le répertoire ˜/*Rapport*/*Docs* en utilisant la commande ***touch*** *nom_fichier*. A quel groupe appartient-il ?

    b. Changer les droits de ce fichier en utilisant la commande ***chmod***.

    c. Supprimez les droits de lecture sur votre fichier. Essayer de l'ouvrir avec un éditeur de texte. Remettez les droits de lecture, mais supprimez les droits d'écriture. Essayez de l'ouvrir avec un éditeur de texte et rajouter une ligne au fichier et essayez de le sauvegarder.

    d. A quoi correspondent les droits de lecture/écriture/exécution pour les répertoires ? Testez en décrivant ce que vous faites.

TP3- Scripts en Shell Unix sh

Exercice 1 : ***Premiers Scripts***

1- Ecrire un script qui permet de faire afficher la date du jour sur une ligne et sur la ligne suivante les usagers branchés au système.

2- Ecrire un script qui permet de demander à l'utilisateur de saisir son nom et lui affiche le message suivant :

Bonjour $nom , votre dossier personnel contient :

Exercice 2 : ***Interactivité du Shell***

Les shells scripts interactifs sont basés sur des commandes shell (***echo*** et ***read***). Créer un script interactif qui permet la conversation suivante entre le système et l'utilisateur :

Cher utilisateur quel est votre nom ?

Ravi de vous connaitre, $nom

Quel est votre prénom et votre âge?

Vous avez $age ans $prenom $nom

Exercice 3 : ***Arguments***

Écrivez un programme Shell qui affiche un à un les arguments de la commande. Par exemple :

$ ./argument un deux trois quatre

4 arguments :

1 - un

2 - deux

3 - trois

4 - quatre

Exercice 4 : ***Les opérations***

1- Écrivez un script qui accepte 2 chiffres en paramètre, additionne ces deux chiffres et affiche le résultat.

2- Ecrivez un script qui permet de retourner le nombre de caractère d'une chaîne de caractère donnée en paramètre.

Exercice 5 : ***test et instruction conditionnelle***

Ecrivez un script qui permet de lire le nom d'un fichier. Le script doit afficher le contenu si le fichier est ordinaire ou bien l'ouvrir si c'est un répertoire.

Exercice 6 : *L'instruction case*

Ecrivez un script "*supprime*" qui pose la question "*Voulez-vous supprimer le fichier nom_fichier ?*" et qui donne les résultats suivants après exécution :
- Si la réponse est oui, Oui, OUI , O il doit afficher le message suivant :*Votre fichier est supprimé.*
- Si la réponse est non, NON, N, n le programme retourne le message suivant :*Votre fichier n'a pas été supprimé*.
- Sinon le programme doit afficher le message suivant : *Réponse incorrecte.*

Exercice 7: Les boucles while et until

1- Ecrivez un script qui permet de tester si un mot de passe saisi par l'utilisateur est bien "ubuntu " en donnant à l'utilisateur la main de saisir le mot quatre fois. Donnez une version en utilisant le boucle **while** et une autre avec l'instruction **until**.

2- Ecrivez un script qui convertit les minutes passées en paramètre en secondes. Le programme doit tester si l'utilisateur a bien donné les minutes en argument.

Exercice 8 : *La boucle for*

1- Écrirez un programme Shell qui compte de 1 à 10 en utilisant la boucle for.

2- Ecrivez un script qui explore un répertoire donné en paramètre et donne le type de chaque fichier rencontré (fichier ordinaire, répertoire, fichier en mode bloc ou fichier en mode caractère). Modifiez le script pour qu'il puisse prendre plusieurs répertoires en paramètre.

3- Ecrivez un script qui teste si des chaînes de caractères passées en paramètre appartiennent à un ou plusieurs fichiers d'un répertoire lui aussi passé en paramètre.(***boucles for imbriquées***) .

Exercice 9 : La boucle while et redirection des entrées /Sorties

1- Ecrivez un script qui lit un fichier donné en paramètre et affiche chaque ligne de ce fichier précédée de son numéro de ligne dans le fichier.

2- Le fichier précédent contient les informations d'un client tel que chaque ligne correspond à un client et comporte son nom, son prénom et son age. Ecrivez un script qui lit ce fichier ligne par ligne et afficher la phrase suivant pour chaque ligne :

« Le client numéro *Num_client_dans_fichier* est agé de *age_client* est appelé *prenom_client,nom_client* »

Exercice 10 : *Les fonctions.*

1- Ecrivez une fonction *effacer_fichier()* qui permet de confirmer ou non la suppression d'un fichier donné en paramètre à la fonction.

2- Ecrivez un script *Suppresion* qui permet de supprimer les fichiers *.log* et *.txt* d'un répertoire donné en paramètre au script en utilisant la fonction *effacer_fichier().*

Exercice 11 :

Ecrivez un script qui prend en paramètre le nom d'un processus (programme en cours d'exécution) cherche son PID et le tue. Le programme doit tester si l'utilisateur a bien passé le nom de processus en paramètre.

Exercice 12:

Ecrivez un script *profondeur* qui fait un parcours en profondeur d'un répertoire qui affiche le nom des fichiers rencontrés et les fils de chaque répertoire rencontré.

Projet Unix SMI S3 2015-2016 : Agenda

1- Descriptif du projet

Le projet consiste à réaliser, à l'aide de scripts shell sh une application de gestion des rendez-vous d'un cabinet médical. Le script final doit permettre :

1- Enregistrement et authentification de plusieurs utilisateurs.

2- Enregistrements des patients.

3- Gestion des rendez-vous.

L'interface sera entièrement textuelle.
L'utilisateur du programme doit s'identifier avant d'utiliser l'application. Un utilisateur est décrit par les éléments suivants (stockés dans un fichier) :

1. Son nom
2. Mot de passe
3. Sa description

2- Descriptif technique

a. Démarrage

Le script du projet devra se nommer GestionRendezVous.sh.

b. Structure de données

Le projet sera modélisé sous forme de fichiers et de répertoires. Il doit utiliser des scripts shell uniquement. Pour simplifier l'écriture du script shell qui englobe l'ensemble du projet, vous pouvez écrire quelques scripts utilitaires effectuant des parties du travail.

c. IHM

L'application étant en mode texte. La communication utilisateur / programme sera à base de commandes en mode texte. Un menu contenant les options de l'application permettra de :

Menu 1 : Gestion des Utilisateurs

1- Enregistrer Un nouvel utilisateur

2- S'Authentifier

3- Quitter

Une fois l'utilisateur connecté, le menu suivant s'affiche :

Menu 2 : Gestion des rendez-vous

1. Ajouter un patient.

2. Prendre un Rendez-vous.

3. Consulter les Rendez-vous.

4. Consulter les jours fériés.

5. Abandonner.

Pour prendre un rendez-vous, l'utilisateur doit vérifier les disponibilités pour l'horaire demandé par le patient. Un même horaire (entre 09h00 et 10h00 par exemple) peut être affecté au maximum à trois personnes.

L'utilisateur devra à chaque fois vérifier si le jour demandé ne coïncide pas avec des samedis et des dimanches en utilisant la commande « *cal » par exemple.*

Durant l'utilisation une touche spécifique permettra d'afficher le menu2 décrit ci-dessus.

d. Fichiers nécessaires

Le projet doit contenir deux fichiers enregistrant les informations sur les utilisateurs et les patients : *FichierUtilisateur*, *FichierPatient*, *FichierRendezVous*, *FichierJoursFériés.*

1- Le *FichierUtilisateur* contient les informations sur les utilisateurs. Il a la structure suivante :

Identifiant unique de l'utilisateur: login: password :Nom-Prenom :Tel :Adresse

2- Le *FichierPatient* contient les informations sur les patients. Il a la structure suivante :

Identifiant unique du patient:Nom-Prenom:Tel Adresse

3- Le fichier *FichierRendezVous* contiendra les informations des joueurs enregistrés dans le jeu.

Identifiant unique du Rendez-vous:Nom-Prenem du patient:Heure:Date

4- Le fichier *FichierJoursFérié* contiendra les jours fériés à prendre en considération lors des prises des Rendez-vous.

Identifiant unique du Jour Férié:libéllé:Date

3- Rapport & script

Le rapport sera à rendre au format PDF, la page de garde devra contenir le nom des binômes.
Après une rapide introduction, vous donnerez les informations relatives à l'interprétation du sujet et les choix que vous avez effectués.
Les scripts devront être inclus en annexe.
Les scripts devront être commentés.

Un CD contenant les scripts doit être rendu .le dossier sauvegardé doit avoir la nomenclature suivante : *nombinome1_nombinome2.tar.gz*

Quelques Références

1. Shells Linux et Unix par la pratique, par **Blaess, Christophe ,** Editeur: **Eyrolles ,** Publication: **2008 .** ISBN: **978-2-212-12273-2**

2. Scripts shell Linux et Unix - Avec 30 scripts prêts à l'emploi, par : **Blaess, Christophe.** Editeur: **Eyrolles,** Publication: **2012,** ISBN: **978-2-212-13579-4**

3. Linux : The Complete Reference 6th edition, par Petersen et Richard, Editeur: McGraw-Hill USA,

4. Cours Pierre-Yves AILLET : Université de Picardie Jules Verne d'Amiens,