Plan Pédagogique du cours

Module: UNIX/LINUX

Section : informatique

Niveau : 3ème niveau (gestion, industriel, réseau)

Volume Horaire : 22,5 heures Cours Intégrés + 45 Travaux Pratiques

Coefficient : 3

Evaluation : Interrogations Orale - Travaux dirigés – Travaux Pratiques – Interrogations Ecrites

Suivi des versions du support

Version	Date	Rédigé Par	Raison
1.0	Juin 2004	Tahar Haouet	Création du document
2.0	Février 2005	Tahar Haouet	Non correspondance charge horaire réalisé/ estimé Révision et Amélioration globale du support

Objectif général du cours

- Découvrir les spécificités du système Unix (Architecture- noyau…)

- Devenir autonome pour une première prise en main d'un système Unix

- Découvrir le système de gestion de fichier

- connaître les mécanismes de gestion des processus sous Unix

- Acquérir la connaissance des commandes fondamentales du système d'exploitation Unix à travers des exercices modulaires de difficulté progressive

- Ecrire des scripts Shell (BASH)

Table des matières

LEÇON 1 : PRESENTATION D'UNIX 5
1-1- Historique 5
1-2 Les fonctions principales 6
1-3 Caractéristiques générales d'Unix 7
1-4-Architecture du système 9
1-5- Conclusion 10

LEÇON 2 : CONNEXION D'UN UTILISATEUR 13
2-1- Connexion au système : 13
2-2- Interface de commande Shell 14
2-3 Environnement de l'utilisateur 17
2-4 Utilisateurs et groupes d'utilisateurs 18

LEÇON 3 : LE SYSTEME DE FICHIERS UNIX 22
3-1- Le système de fichiers 22
3-2- Désignation des fichiers 22
3-3- Types de Fichiers 24
3-4 inode 24
3-5 Organisation des disques System V 27

LEÇON 4 : PRINCIPALES COMMANDES SUR LES FICHIERS UNIX 29
4-1- Principales Commandes sur les répertoires 29
4-2- Principales Commandes sur les Fichiers 29
4-3 Copie de fichiers (cp) 31
4-4 Déplacement et suppression de fichiers (mv, rm) 32
4-5 Les Liens 32
4-6 Quelques utilitaires (diff, du, wc) 33
4-7- Recherche de fichiers (find): 34

LEÇON 5 : DROITS D'ACCES AUX FICHIERS UNIX 38
5-1. Protection des fichiers 38
5-2 Modification des droits d'accès aux fichiers 39
5-3. Les droits spéciaux : 40

LEÇON 6 : GESTION DE PROCESSUS 43
6-1 Notion sur les processus 43
6-2 Etats d'un processus 44
6-3 Priorité d'un processus 45
6-4 Modes d'exécution d'une commande: 45

LEÇON 7 : COMPOSITION DE PROCESSUS 47
7-1- La redirection des entrées sorties 47
7-2- La composition de processus. 48

LEÇON 8 : COMMANDES DE GESTION DE PROCESSUS 51
8-1-Commandes de gestion de processus 51
8-2 Le job control 53

LEÇON 9 : LES FILTRES 55
9-1- Définition 55
9-2- Filtre grep 56
9-3- Filtre sort 57

9-4-Filtre cut _____57
9-5- Filtre tr _____58
9-6 L'utilitaire sed _____58

LEÇON 10 : PROGRAMMATION SHELL _____62
10-1- Introduction : _____62
10-2- Programmation de base en shell _____63

LEÇON 11 : PROGRAMMATION SHELL (STRUCTURES DE CONTROLE) _____66
11-1- les structures de contrôle: _____66
11-2- Les instructions conditionnelles: _____66
11-3- Les itérations _____68
11-4 Arithmétique entière sur les variables. _____70

Leçon 1 : Présentation du système Unix

Prérequis :

Système d'exploitation, Architecture des ordinateurs

Objectif du chapitre :

Présenter et introduire le système Unix

Durée : 1h30

Eléments du Contenu :

Connaître les origines du système, ses fonctionnalités et son architecture

LEÇON 1 : PRESENTATION D'UNIX

1-1- Historique

L'histoire d'UNIX débute dans les années 60 et peut être résumée de la façon suivante :

- **1969** : premier système Unix écrit par Ken Thompson aux Bell Labs d'AT&T, en assembleur sur PDP-7, puis en langage "B". A la base appelé Unics (jeu de mot formé à partir de Multics, gros système développé fin 60 par le MIT et General Electric)
- **1973** : Dennis Ritchie, qui a inventé le langage "C", récrit Unix en C avec Thompson (vers.4). Rend possible et entraîne le portage d'Unix sur de nombreuses machines (sources d'Unix distribués à de nombreuses universités et sociétés commerciales). Une 3e personne, Brian Kernighan, contribue également fortement aux premiers développements d'Unix.
- **1976** : publication de la version 6 du Manuel Unix ("manuel du programmeur")
- **1977** : divergence en 2 grandes familles Unix :
    - AT&T : System III, puis **System V** (1990 : release 4, abrégé SVR4)
    - Uni. de Californie à Berkeley : **BSD** (Berkeley Software Distribution, act.vers.4.4), originalités : vi, propre système de fichier, C-shell, gestion virtuelle mémoire, job contrôle, liens symboliques, TCP/IP...
- **1980** : Bjarne Stroustrup (Bell Labs) définit le langage C++ (extension "objet" du C)
- **1987** : apport du MIT en matière de graphique et fenêtrage : X-window version 11 (act. release 6, abrégé X11R6). Puis formation de 2 Consortiums rivaux :
    - **OSF** (Open Software Fundation) (DEC, HP, IBM...) : fenêtrage **Motif**...
    - Unix International (AT&T, Sun...) : système de fenêtrage OpenLook/OpenWindows
- **1990** : création par AT&T de USL (Unix System Laboratories) qui reprend les activités Unix
- **1993** : AT&T vend USL à Novell (juin) qui donne ensuite les droits "Unix" à l'organisation de standardisation X/Open (octobre)
- **1993** : réunification des familles Unix (sous la pression de la concurrence de "Windows NT" de Microsoft !) : convergence des constructeurs vers Unix SVR4 qui se dote de la plupart des

extensions BSD, standardisation appels systèmes (POSIX), environnement de bureau CDE (Common Desktop Environment). Nouveau rôle de l'OSF. Ralliement de Sun à Motif puis à l'OSF.

- **1994** : Linux, Unix gratuit pour plateforme Intel 486 (Linus Torvalds)

1-2 Les fonctions principales

UNIX est un système d'exploitation dont voici les tâches principales :

Partage des ressources équitables

UNIX veille à ce que toutes les ressources de l'ordinateur (imprimante, mémoire, ...) soient partagées équitablement entre tous les processus.

Par exemple si vous travaillez sur une application du genre base de données, vous lancez une requête (commande dans le langage base de données) coûteuse en temps, pour patienter rien ne vous empêche de vous lancer un PacMan. Vous vous retrouvez donc avec deux process lancés en même temps, c'est le système d'exploitation qui est chargé de faire en sorte que les deux process puissent utiliser les ressources de manière équitable et que le deuxième process lancé n'attende pas la terminaison du premier pour se lancer.

Le fait de pouvoir exécuter plusieurs process ou tâches en même temps, en parallèle, est appelé multitâches. UNIX est multitâches.

Système multitâche et Multi-Utilisateur

Un système d'exploitation multitâche attribue périodiquement (quantum de temps de l'ordre du centième de seconde) l'UC à une tâche (exécution d'un programme) différente dans le but de faire progresser l'exécution de plusieurs programmes à la fois.

- La notion d'ordonnancement de tâches est alors apparue.
- L'utilisateur a l'impression que plusieurs programmes sont exécutés « simultanément ».
- Un système multitâche est aussi appelé système à temps partagé.

A contrario, un système d'exploitation monotâche exécute une commande uniquement lorsque la précédente est terminée.

Un système multitâche peut permettre à plusieurs utilisateurs de travailler simultanément, il est alors dit multi-utilisateur.

- Le système d'exploitation alloue chaque quantum de temps à des programmes de différents utilisateurs.
- Ainsi tous les utilisateurs ont l'impression de travailler simultanément.

Interface avec le matériel

UNIX par définition des systèmes d'exploitation fait en sorte qu'aucun process accède directement à une ressource matérielle (disque dur, lecteur de disquette,...). Pour accéder à ces ressources on passe par l'intermédiaire de fichiers spéciaux, un fichier spécial est vu pour un utilisateur comme un fichier classique, pour écrire sur une disquette dans le lecteur de disquette, on n'a qu'à écrire dans le fichier spécial du lecteur de disquette. De même pour lire dans un disque dur, on va lire le fichier spécial du disque dur.

Gestion de la mémoire

Il existe deux types de mémoire, la mémoire volatile et la mémoire statique, quand on éteint et rallume l'ordinateur, toutes les données présentes dans la première ont disparu, et les données dans la seconde sont toujours présentes. Concrètement la mémoire volatile se trouve dans la RAM, la mémoire statique dans le disque dur. Dans le vocabulaire Unix, quand on parle de mémoire on sous entend mémoire volatile ou RAM, c'est la convention qui sera adoptée pour la suite du cours.

Tout programme qui s'exécute, ou process, a besoin de mémoire pour y stocker notamment les données qui manipulent. Malheureusement l'ordinateur dispose généralement d'une quantité de mémoire limitée et non extensible. UNIX doit donc faire en sorte que la mémoire soit bien partagée entre tous les process, un process ne doit pas s'accaparer toute la mémoire, sans quoi les autres process ne pourraient plus fonctionner.

La mémoire est vue comme une ressource matérielle, UNIX doit donc vérifier qu'aucun process accède à la mémoire directement ou ne se réserve une zone de la mémoire.

Gestion des fichiers

UNIX fournit les outils nécessaires pour stocker les données et pour pouvoir les récupérer rapidement et facilement. Il fournit les outils pour pouvoir visualiser l'ensemble des fichiers de manière simple. Ces fichiers se trouvent sur le disque dur, on nomme cela un système de fichiers ou File System en anglais.

UNIX fournit, en outre, un mécanisme de protection des fichiers. Plusieurs utilisateurs peuvent travailler en même temps sur la même machine, c'est la notion de multi-utilisateurs. Chaque utilisateur du système dispose de ses fichiers, UNIX lui donne le moyen de protéger ses fichiers, et d'accorder le droit ou non à d'autres utilisateurs d'accéder à ses fichiers.

1-3 Caractéristiques générales d'Unix

Caractéristiques :

- Le noyau Unix est écrit à plus de 95% en C, langage de haut niveau. Moins de 50% de ce code dépend de la machine. Ceci a pour conséquence une portabilité accrue. Unix est parmi les rares systèmes aujourd'hui disponibles sur tous types d'ordinateurs, du PC au Mainframe.

- Unix est le standard de facto des **systèmes ouverts**, c'est-à-dire des systèmes dont les spécifications sont publiques (Non propriétaire).
- Unix est au centre des efforts de standardisation et de normalisation qui ont pour objet de définir des interfaces standard pour les systèmes d'exploitation pour faciliter les développements et l'interopérabilité entre des machines hétérogènes.
- La plupart des développements dans les domaines de pointe (CAO, bases de données objets, Internet, etc.) se font aujourd'hui majoritairement sur plate-forme UNIX.
- Disponible sur toutes les plateformes, y compris PC (Linux, FreeBSD...)
- interactif, temps partagé, mais offrant aussi des extensions pour le "temps réel"
- multi-tâches et Multi-Utilisateur
- compatibilité totale entre fichiers et périphériques
- multiprocesseurs symétriques, multi-threaded
- facilement extensible et modifiable (ajout de commandes...)
- "case sensitive", c'est à dire qu'il distingue les majuscules des minuscules (tant au niveau des commandes/options que des noms de fichiers) contrairement à VMS ou MS-DOS ("ls -l" n'a pas même effet que "ls -L")
- grande richesse d'outils et fonctionnalités (résultant d'un effort de développement collectif, contrairement aux autres systèmes d'exploitation)
- Compilateurs et outils de programmation et déboguage (C / C++ / Perl)
- L'accès aux réseaux et à ses ressources
- Sécurité contre les accès non autorisés

Ses points forts

- Très forte portabilité des programmes ; il existe près de 3000 plates-formes UNIX, du PC à l'ordinateur central. Un programme développé sur une de ces plates-formes peut être facilement porté sur une autre plate-forme UNIX, souvent par simple recompilation.
- Interface utilisateur simple (shell).
- Un système de fichiers hiérarchique et arborescent.
- Une structure simple des fichiers (suite de caractères).
- Interface simple et homogène pour tous les périphériques : *tout est fichier*, y compris les périphériques qui sont traités comme des

fichiers spéciaux. Cette philosophie du "tout fichier" rend très simple la redirection des flux d'entrées/sorties entre les processus.

Ses points faibles

- Mal adapté aux applications dites "temps réel" ; ces applications ont besoin d'avoir un temps de réponse minimal garanti par exemple les applications dites "embarquées", qui contrôlent une navette spatiale ou un missile. Mais des efforts importants ont été réalisés pour concilier UNIX avec le temps réel.
- Suppose encore des utilisateurs avertis : une station Unix n'est pas un Macintosh ou un PC. Le monde UNIX n'est toujours pas destiné au grand public. Il reste cantonné pour l'instant dans le monde de l'ingénierie (station de travail) ou des serveurs. Cela dit, un grand effort a été accompli récemment pour simplifier l'utilisation des machines UNIX à travers des interfaces graphiques qui offrent une convivialité appréciable.
- Les produits sous UNIX restent plus chers et plus rares que ceux du monde de la micro informatique.
- Malgré les efforts de standardisation, les problèmes d'inter-opérabilité entre les différents UNIX du marché persistent. Le label UNIX 95 est un pas important qui a été franchi pour résoudre ce problème. Mais très peu d'UNIX du commerce ont pour l'instant reçu ce label.

1-4-Architecture du système

Unix est un système d'exploitation, constitué du noyau Unix, d'un interpréteur de commandes et d'un grand nombre d'utilitaires

Le noyau assure la gestion des ressources physiques (processeurs, mémoires, périphériques) et logiciels (processus, fichiers…). L'interface entre les programmes des utilisateurs et le noyau est défini par un ensemble de procédures et de fonctions, soit directement au niveau du noyau, soit par l'intermédiaire de bibliothèques.

Pour ce qui concerne l'architecture du noyau proprement dit, les éléments caractéristiques sont les suivants :

Le noyau est constitué d'un ensemble de procédures et de fonctions écrites en langage C.

L'utilisateur d'Unix n'accède pas directement au noyau mais à un interpréteur de commandes : le shell

Il existe plusieurs shell différents sous Unix

Structure du système UNIX

<table>
<tr><td colspan="2">Applications</td></tr>
<tr><td colspan="2">Appels Systèmes Unix</td></tr>
<tr><td rowspan="2">Gestion de processus</td><td>Système de gestion de Fichiers</td></tr>
<tr><td>Drivers Périphériques</td></tr>
<tr><td colspan="2">Contrôle Matériel</td></tr>
<tr><td colspan="2">Matériel</td></tr>
</table>

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

On a vu auparavant que les process ne pouvaient pas accéder directement aux ressources matérielles, en fait les process passent par le noyau pour y accéder, pour cela ils disposent d'un ensemble de commandes appelées " appels système " 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.

Pour faire marcher l'ordinateur, l'utilisateur dispose des logiciels ou d'un utilitaire qui lui permet la saisie directe de commandes. On appelle cet utilitaire le shell (coquille en français). Son rôle est d'interpréter les commandes de l'utilisateur avant transmission au noyau, c'est pourquoi on parle aussi d'interpréteur de commandes. On trouve l'équivalent sous DOS qui peut être considéré comme un shell.

Il existe plusieurs types de shell, ils se différencient par la syntaxe et la richesse des commandes. Le plus commun est le Bourne-Shell, on trouve aussi le C-Shell qui s'apparente au langage de programmation C, le Korn Shell, le Posix Shell, et sous Linux le bash-shell.

1-5- Conclusion

Unix est un système d'exploitation multitâche et Multi-Utilisateur structuré en couches, garantissant sa portabilité (existence sur presque toutes les plateformes matérielles).

Le noyau Unix contient les principales fonctionnalités du système.

La bibliothèque C fait l'interface entre le noyau et les programmes utilisateurs.

Unix fournit une interface homme/machine très puissante appelée shell mais aussi des interfaces graphiques.

Unix fournit de nombreux outils (en standard et dans le domaine public).

Leçon 2 : Connexion d'un utilisateur

Prérequis :

Chapitre précèdent

Objectif du chapitre :

Devenir autonome pour une première prise en main d'un système Unix

Durée : 2 Séances (2x 1h30)

Eléments du Contenu :

Connexion au système - Interface de commande Shell – Personnaliser la session – fichiers d'initialisation - définition des utilisateurs et des groupes

LEÇON 2 : CONNEXION D'UN UTILISATEUR

2-1- Connexion au système :

2-1-1 Ouverture de session

Avant de tenter une connexion, il faut d'abord vous assurer que vous ayez 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.

Tout Utilisateur est identifié par un (**login name**) et ne peut utiliser le système que si son nom a préalablement été défini par l'administrateur du système (root)

Pour qu'un utilisateur puisse travailler sur un système Unix, il doit s'identifier en indiquant son nom (**login**), puis son mot de passe (**passwd**)

Apparaissant alors à l'écran un certain nombre d'informations (informations générales, date, arrivée de messages, date de dernière connexion) Puis le système lance un programme qui est généralement un interpréteur de commandes.

A partir de ce moment l'utilisateur est connecté (il est entré en session).

A la mise sous tension, vous pouvez aussi disposer d'une interface graphique de connexion, au lieu d'avoir un simple login: avec le curseur qui clignote, vous avez une fenêtre ou bannière qui vous invite à saisir votre login et votre mot de passe. C'est notamment le cas pour la configuration par défaut de Redhat Fedora Core mais aussi pour les versions récentes de UnixWare.

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 « $ » (appelé prompt) suivi du curseur qui clignote. 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 fenêtré avec utilisation de la souris, où il vous sera possible de lancer un shell pour pouvoir taper des commandes UNIX.

2-1-2 Changement de password

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. Certains UNIX font en sorte que vous ne puissiez pas saisir un mot de passe simple, il faudra mettre au moins 6 caractères, avec au moins un ou deux, caractère non alphabétique.

Quand vous saisissez votre mot de passe, il ne paraît pas en clair, aussi par précaution, le système vous demande de le saisir deux fois.

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 débloquer.

2-1-3 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 avec une commande Exit, ou Logout, qui a strictement le même effet.

2-2- Interface de commande Shell

Un Shell (Interprète des commandes) est un programme qui accepte des commandes textuelles et les exécute.

L'interpréteur de commandes ou shell est l'interface entre l'utilisateur et le système d'exploitation. C'est à la fois un langage de commandes et un langage de programmation. Son rôle est triple :

- lire les commandes à partir du clavier ou de toute autre source,
- les exécuter,
- fournir si nécessaire un flux de sortie.

Les shells ne font pas partie du noyau d'Unix, mais sont des services offerts "en plus". En fait Unix est conçu pour être "attaqué" depuis des programmes écrits en C, langage développé à cette fin.

Il existe deux grandes familles de shells sur Unix. L'une contient sh et ksh, l'autre csh et tcsh.

•sh, dit Bourne Shell, est un shell assez ancien et très répandu.

•ksh, dit Korn Shell, est une évolution de sh, avec des possibilités supplémentaires.

ksh a récemment été retenu comme "norme" par un consortium Unix ;

•csh, dit C-Shell, est le préféré des développeurs en C, car ses commandes ont une syntaxe très voisine de celle de ce dernier langage ;

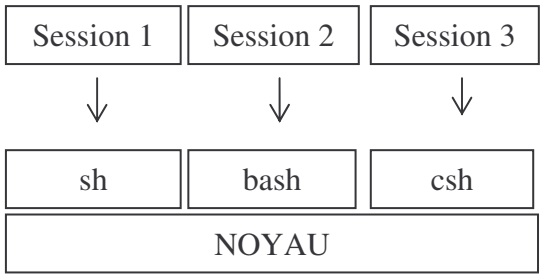
•tcsh est une évolution de csh. Elle offre la complétion des noms de commandes, des noms de variables et des noms de fichiers.

Comme ksh est une nette amélioration de sh, il est recommandé de l'utiliser à la place de ce dernier. Les différences entre tcsh et csh offrent surtout un agrément à l'utilisateur, comme le rappel des commandes précédentes avec les touches curseur du clavier et la complétion des commandes.

Un utilisateur peut avoir différentes sessions ouvertes, chacune ayant son propre shell.

Un Shell est un processus qui s'exécute sur la machine.

Nous verrons les processus plus en détail aux chapitres suivants



2-2-1- Caractéristiques du Shell

Un Shell se caractérise par un ensemble de commandes internes et un ensemble de variables d'environnement.

Une commande interne est une commande contenue dans la boîte à outils du Shell, elle est donc littéralement intégrée. C'est soit pour des raisons de performance, soit parce qu'elle à besoin d'un accès direct aux variables d'environnement du Shell

Les commandes internes s'exécutent plus rapidement que les commandes externes, qui nécessitent habituellement de Créer un processus (tâche)

Les variables d'environnements du Shell sont des variables qui contiennent des informations nécessaires pour l'exécution correcte du shell. A chaque ouverture de session le Shell initialise automatiquement ces variables.

Shell	commandes internes
Bash	50
Boume shell	32
C-Shell	52

Nom de variable	Fonction
EDITOR	Chemin d'accès de votre éditeur de texte
HOME	Chemin d'accès de votre répertoire d'accueil
LOGNAME	Identifiant d'utilisateur utilisé pour le login
MAIL	Chemin d'accès du programme de courrier électronique
PATH	Liste des répertoires à par courir pur trouver un programme
SHELL	Chemin d'accès du programme Shell
TERM	Type de terminal utilisé
USER	Identifiant de l'utilisateur logué

2-2-2- Changement du Shell

Si vous souhaiter changer le Shell, vous devez utiliser la commande chsh. La syntaxe de cette commande est :

```
chsh identifiant nom_du_shell
```

Nom_du_shell : Le chemin complet du répertoire contenant ce shell.

Shell	Chemin d'accès au Shell
Bash	/bin/bash
Bourne shell	/bin/sh
C-shell	/bin/csh
Korn shell	/bin/ksh
Tcsh	/bin/tcsh

Exemple :

```
chsh farhat /bin/ksh
```

2-2-3- Structures des commandes Shell

Les commandes des shells Unix ont typiquement la forme suivante :

```
nom_de_la_commande [option(s)] [arguments(s)]
```

Les options sont introduites en général par un caractère "-", parfois par un "+".

Elles servent à régler finement l'effet (résultat) des commandes

Les arguments spécifient les objets (variables ou fichiers) sur lesquels la commande va s'appliquer.

Les crochets autour des arguments et des options signifient qu ceux si sont optionnels.

Il existe trois types de commandes dans les shell Unix :

•Certaines sont **prédéfinies** par le shell considérée, comme *time* :

•D'autres dont des **fichiers exécutables**, obtenus par développement avec par exemple un langage d'assemblage, C ou encore C++.

•D'autres enfin dont des scripts, c'est à dire des fichiers texte contenant des commandes d'un des shells. On peut ainsi programmer pas mal de choses dans Unix.

A noter que les shells Unix font la différence entre les lettres minuscules et majuscules contrairement à Windows.

2-2-4- Exemples de commandes Shell :

- date et heure courants

```
$ date
Tue Nov     4     13 :39 :24 EST 1997
```

- liste les utilisateurs connectés au serveur. (utilisateur courant)

```
$who am i
```

```
farhat      tty03      Nov  6     08:26
```

- commande bidon

```
$dare
dare: commande not found
```

- liste tous les fichiers du répertoire courant (-a) all

```
$ls –a
.       .exrc           ch1         ch2         intro
..      .profile  ch10      ch3         mp3
```

- Affiche le répertoire de travail

```
$pwd
/users/abdul/
```

- changer de répertoire

```
$ cd  /home/farhat
```

2-3 Environnement de l'utilisateur

Qu'est-ce qu'un environnement ?

Un environnement est un ensemble de variables et de fichiers de configuration contenant des états et des options du système. Par exemple, une variable d'environnement peut indiquer que votre terminal est un terminal VT100, une autre que votre shell préféré est bash, et une dernière que vous maintenez un système de manuel personnel qu'il faut consulter lorsque la commande man est invoquée.

Variables

Le système utilise des variables comme :

TERM type de terminal courant

PATH chemin de recherche des commandes

LD LIBRARY PATH chemin de recherche des bibliothèques partagées

MANPATH chemin de recherche des manuels

EDITOR éditeur de texte par défaut

Certaines applications nécessitent également des variables d'environnement pour fonctionner. On peut les afficher indépendamment avec la commande **echo $VAR**, si la variable s'appelle **VAR**.

On peut afficher une liste des variables avec les commandes env, ou set.

On peut modifier les variables d'environnement avec les commandes export ou set

Notez bien que ces modifications sont temporaires (jusqu'`a la fin du shell courant). Pour les rendre permanentes, il faut les ajouter aux fichiers de démarrage (initialisation) du shell (voir section suivante).

Commandes de configuration

La plupart des commandes de configuration sont exécutées au login. D'autres commandes sont exécutées à chaque lancement d'un shell. Parmi celles-ci, citons les commandes suivantes :

stty pour initialiser les paramètres du terminal.

mail pour vérifier si vous avez du courrier privé.

quota pour contrôler si vous avez des dépassements de quota.

Fichiers d'initialisation

Lorsqu'un utilisateur se connecte sur un système UNIX, certaines tâches sont exécutées automatiquement. Ces tâches sont définies dans les fichiers d'initialisation se trouvant dans le répertoire principal de chaque utilisateur.

La plupart de ces fichiers commencent par « . » et ils ne sont donc pas visibles avec ls, mais avec ls -a.

.bash_profile Ce fichier contient des séquences d'instructions exécutées à chaque lancement d'un shell de connexion bash.

.bashrc Ce fichier contient des séquences d'instructions exécutées à chaque nouveau sous-shell bash ouvert (excluant les shells de connexion).

.bash_logout Ce fichier contient des commandes exécutées en fin de connexion pour un shell de connexion bash.

Modifier son environnement

Modifier son environnement de façon permanente se fait toujours dans un fichier de configuration exécuté à l'entrée dans le système ou à la création du shell (.bash_profile ou .bashrc). Pour changer une variable, il faut utiliser l'assignation de variables, avec set ou export.

On pourra toujours quitter en cas de problèmes et reprendre la copie que vous n'aurez pas manqué de faire !

2-4 Utilisateurs et groupes d'utilisateurs

Sur un système UNIX, on trouve deux types de personnes, celle qui va utiliser le système dans le but de produire quelque chose, le système UNIX est pour elle un moyen, un outil. Cette personne est l'utilisateur UNIX, on peut trouver dans cette catégorie, le programmeur, l'utilisateur de base de données, etc.

La deuxième catégorie de personnes est chargé de l'installation, de la configuration et de la bonne marche du système UNIX, ce sont les administrateurs systèmes UNIX.

Sur un système UNIX, les utilisateurs UNIX ont des droits limités, c'est à dire que certaines commandes leurs sont interdites et ils n'ont pas accès à certaines parties du système. Les administrateurs systèmes ont par contre tous les droits sur le système.

Généralement sur un système UNIX, on limite volontairement le nombre d'administrateur (appelé ROOT ou super utilisateur).

Les opérations possibles sur les comptes des utilisateurs sont :

Création, Modification, destruction

Les comptes utilisateurs sont crées par l'administrateur (root) Ces utilisateurs sont rassemblés par **groupes**. Les utilisateurs d'un même groupe peuvent avoir des accès particuliers sur des fichiers appartenant aux usagers du même groupe. En général un groupe rassemble des utilisateurs qui travaillent sur le même projet ou dans un même département…

Le fichier /etc/passwd

Les Utilisateurs du système Unix sont définis dans le fichier /**etc**/**passwd**

Le fichier /**etc**/**passwd** contient toutes les informations relatives aux utilisateurs (login, mots de passe, ...). Seul le superutilisateur (root) doit pouvoir le modifier. Il faut donc modifier les droits de ce fichier de façon à ce qu'il soit en lecture seule pour les autres utilisateurs.

Ce fichier possède un format spécial permettant de repérer chaque utilisateur, chacune de ses lignes possède le format suivant:

```
login :mot_de_passe :UID :GID :HomeDir :shell
```

Login : login de l'utilisateur

Mot_de_passe : mot de passe de l'utilisateur

UID : user identifier : identifiant de l'utilisateur

GID : Group Identifier : identifiant du groupe de l'utilisateur

HmeDir : répertoire home de l'utilisateur

Shell : shell par défaut de l'utilisateur.

Voici un exemple de /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:daemon:/sbin:/bin/bash
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14::/var/lib/uucp/vnc_config:/bin/bash
farhat:x:500:100:Cool......:/home/farhat:/bin/bash
```

Le fichier /etc/group

Le fichier /**etc**/**group** contient la liste des utilisateurs appartenant aux différents groupes. En effet, lorsque de nombreux utilisateurs peuvent avoir accès au système, ceux-ci sont fréquemment rassemblés en différents groupes ayant chacun leurs propres droits d'accès aux fichiers et aux répertoires.

Il se compose de différents champs séparés par « : »

```
nom_de_groupe : champ_special : gid : membre1,membre2
```

Le champ spécial est fréquemment vide.

Le numéro de groupe (gid) est le numéro qui fait le lien entre les fichiers /etc/group et /etc/passwd

Voici un exemple de fichier */etc/group*

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:
tty:x:5:
disk:x:6:
lp:x:7:
wwwadmin:x:8:
kmem:x:9:
wheel:x:10:
mail:x:12:cyrus
news:x:13:news
```

Un même utilisateur peut apparaître dans plusieurs groupes. Lorsqu'il se connecte au système, il appartient au groupe spécifié dans le fichier /etc/passwd (le champ GID). Il peut en changer à l'aide de la commande newgrp

Les protections du fichier doivent empêcher sa modification par les utilisateurs non privilégiés.

Pour ajouter un groupe, l'administrateur peut modifier le fichier /etc/group à l'aide d'un éditeur de texte. Il peut également utiliser la commande addgroup ou groupadd (pas toujours présentes).

Pour ajouter un utilisateur à un groupe, il suffit d'éditer le fichier /etc/group et de rajouter ce nom au bout de la ligne en séparant le nom des membres par une virgule.

Pour supprimer un groupe, il suffit d'éditer le fichier /etc/group et d'effacer la ligne correspondante. Mais attention, il ne faut pas oublier de changer dans le fichier /etc/passwd les numéros (GID) du groupe supprimé, si des utilisateurs y appartenaient. Il est également essentiel de chercher les fichiers et répertoires de ce groupe pour le changer (dans le cas contraire les fichiers et répertoires riquent d'être inaccessibles).

Leçon 3 : Le système de fichier Unix

Prérequis :

Système d'exploitation, Architecture des ordinateurs, Chapitres précédents

Objectif du chapitre :

Connaître l'organisation des fichiers sous Unix

Durée : 1h30

Eléments du Contenu :

Le système de fichier d'Unix – structure des fichiers sur le disque - inodes

Désignation des fichiers

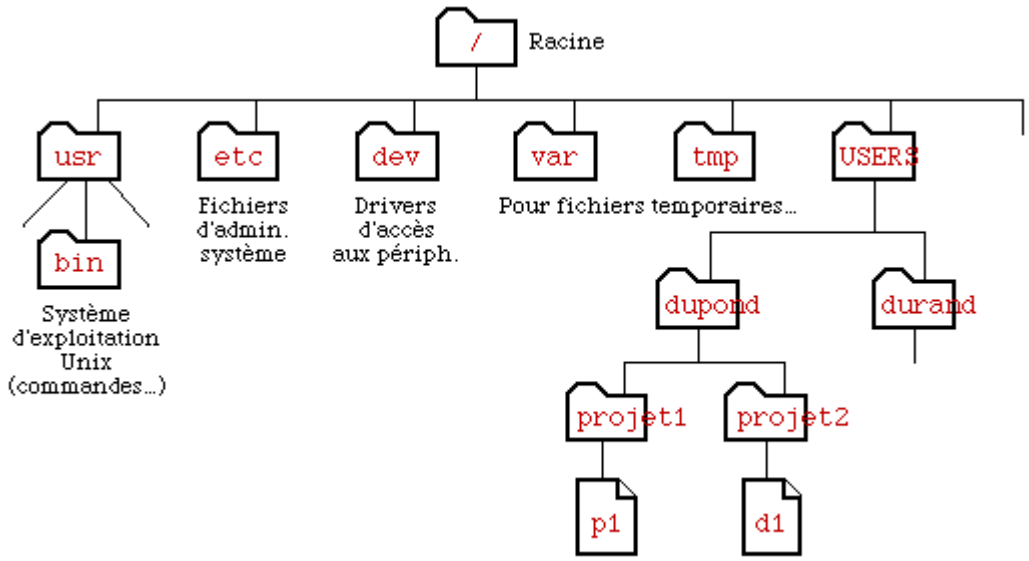
Arborescence des répertoires

Types des fichiers

LEÇON 3 : LE SYSTEME DE FICHIERS UNIX

3-1- Le système de fichiers

Le système de fichier constitue un élément clé d'Unix. Vu par l'utilisateur, le système de fichiers est organisé en une structure arborescente dont les nœuds sont des répertoires et les feuilles des fichiers.



Sous Unix un fichier est :

- Toujours désigné par un nom.
- Possède un unique inode (certaines informations concernant le fichier).
- Possède les fonctionnalités suivantes :
  - ouverture.
  - fermeture.
  - lecture (consultation).
  - écriture (modification)

3-2- Désignation des fichiers

Un fichier est repéré par son nom et sa position dans l'arborescence : son chemin d'accès.(chemin/nom)

Unix fait la différence entre les majuscules et les minuscules (la casse). En théorie tous les caractères du clavier sont autorisés. En pratique, il vaut mieux s'abstenir pour certains (comme * ou ?) . Le caractère - est déconseillé au début d'un nom de fichier (certaines commandes pourraient l'interpréter comme une option).

La syntaxe de ce chemin d'accès est très précise et peut être décrite de deux manières :

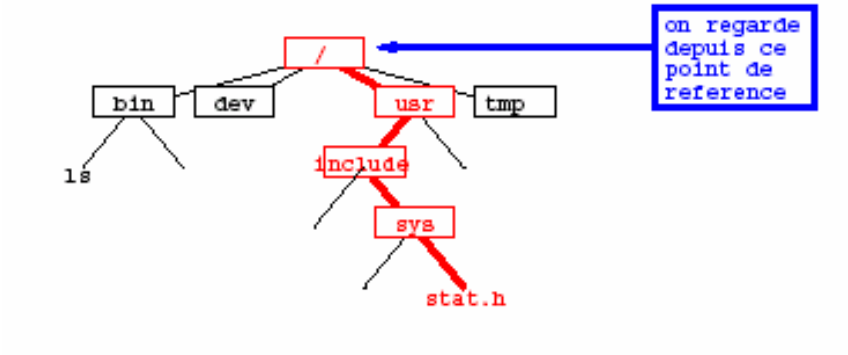
Chemin d'accès absolu :

chemin d'accès = /répertoire/nom

Si le nom de répertoire commence par /, il s'agit d'une référence absolue par rapport au répertoire racine /, constituée d'une liste des répertoires à parcourir depuis la racine pour accéder au fichier.

Il décrit l'itinéraire à emprunter depuis la racine de l'arborescence jusqu'au fichier.

Exemple.



chemin absolu du fichier stat.h

/usr/include/sys/stat.h

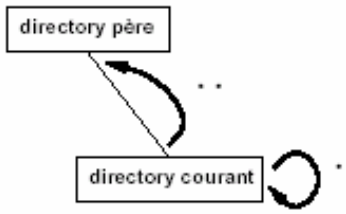
Chemin d'accès relatif

chemin d'accès = répertoire/nom

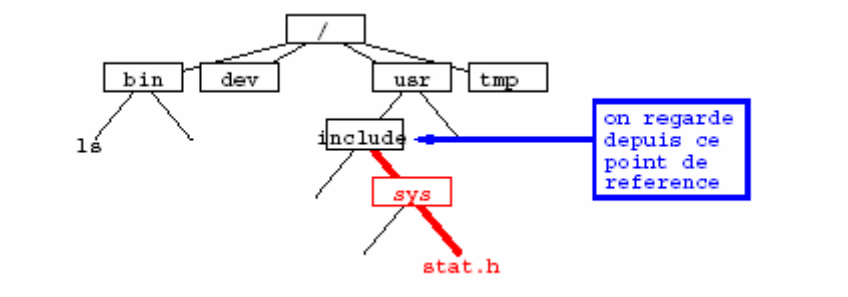
Si le nom de répertoire ne commence pas par /, il s'agit d'une référence relative par rapport au répertoire courant.

Le répertoire courant est noté « **.** »

Le répertoire parent du répertoire courant est noté « **..** »



Exemple



Le chemin relatif du fichier stat.h depuis l'emplacement /usr/include est sys/stat.h

3-3- Types de Fichiers

Le système Unix définit trois types de fichiers :

Les fichiers ordinaires (réguler files) :

Ils servent à mémoriser les programmes et les données des utilisateurs et du système.

Un fichier ordinaire peut contenir du texte, un programme, ou d'autres données. Ca peut être soit un fichier ASCII, avec chacun de ses octets compris entre 0 et 127, chaine7-bit, ou un fichier binaire, ou toutes les possibilités de valeurs de 0 a 255, chaîne 8-bit.

Les fichiers répertoires ou répertoires :

Chaque répertoire contient la liste et la référence des fichiers placés sous son contrôle et la référence du répertoire dont il dépend (répertoire père)

Les fichiers spéciaux :

Ils désignent les périphériques, les tubes ou autres supports de communication inter-processus. Les fichiers spéciaux associés aux périphériques peuvent être caractères (terminaux) ou blocs (disque) ; les entrées/sorties (E/S) se font soit caractère par caractère, soit bloc par bloc, un bloc étant composé de n caractères (512, 1024 ou 2048)

Exemples:

-Fichier ordinaire (-)

-Fichiers répertoires (d)

- Fichier spécial : périphérique accède en mode caractère (c)

- Fichier spécial : périphérique accède en mode bloc (b)

- Tubes (p)

- Lien symbolique (l)

- Sockets (s)

3-4 inode

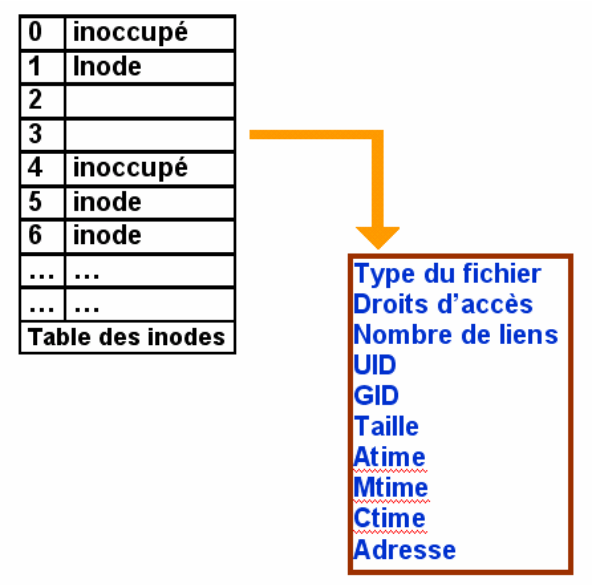
Un Unix File System UFS est toujours muni d'une table des inodes.

Définition

L'Inode ou "nœud d'index " est le passage obligé de tous les échanges entre le système de fichier et la mémoire.

L'inode est la structure qui contient toutes les informations sur un fichier donné à l'exception de sa référence dans l'arborescence (son nom).

Un Inode est créé au même moment qu'un fichier afin de contenir ses informations fondamentales. Tous les inodes sont conservés dans une table, et sont identifiés par un inumber (numéro d'index/numéro d'inode).



Tout fichier possède son unique inode

- L'inode contient la totalité des informations sur le fichier, sauf le nom.
- Les inodes sont tous de même taille.

Les informations des inodes

Les informations stockées dans une inode disque sont :

– utilisateur propriétaire

– groupe propriétaire

– type de fichier

- - fichiers ordinaires
- d répertoire (directory)
- b mode bloc
- c mode caractère
- l lien symbolique
- p pour une fifo (named pipe)
- s pour une socket

– droits d'accès (ugo*rwx)

– date de dernier accès (atime)

– date de dernière modification (mtime)

– date de dernière modification de l'inode (ctime)

– taille du fichier

– adresses des blocs-disque contenant les données du fichier.

Dans une inode en mémoire (fichier en cours d'utilisation par un processus) on trouve d'autres informations supplémentaires :

Le statut de l'inode

```
{
locked,
waiting
inode à écrire,
fichier à écrire,
le fichier est un point de montage
}
```

Et deux valeurs qui permettent de localiser l'inode sur un des disques logiques :

Numéro du disque logique

Numéro de l'inode dans le disque

cette information est inutile sur le disque (on a une bijection entre la position de l'inode sur disque et le numéro d'inode).

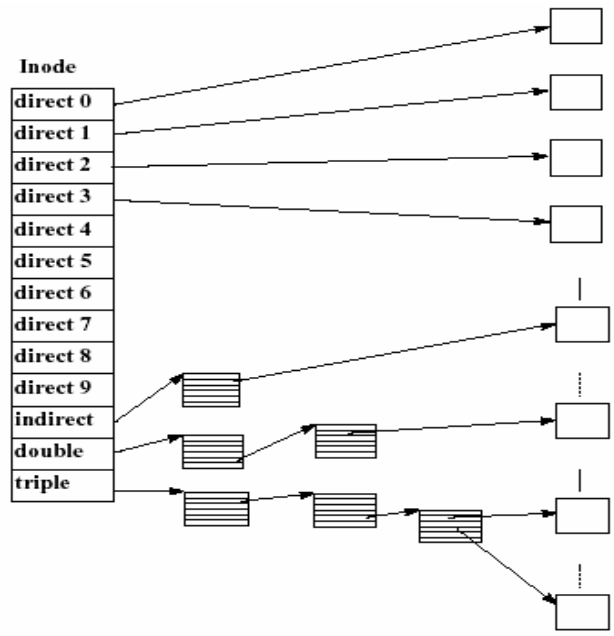
Remarque :

Un fichier peut avoir plusieurs liens donc plusieurs noms. Il suffit pour cela qu'ils pointent sur le même inode(celui du fichier).

Adressage des blocs de données

Une liste de 13 adresses :

- Direct pour les 10 premières adresses.
- Indirect pour la onzième entrée.
- A deux niveaux d'indirection pour la douzième entrée.
- A trois niveaux d'indirection pour la dernière entrée.



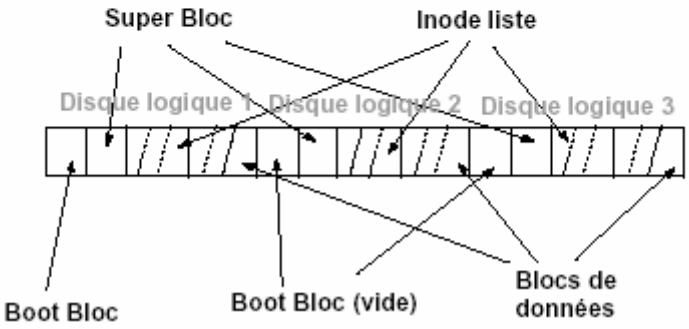
Un bloc de données peut contenir jusqu'à 128 adresses

3-5 Organisation des disques System V

L'organisation disque décrite sur la figure suivante est la plus simple que l'on peut trouver de nos jours sous UNIX, il en existe d'autres où l'on peut en particulier placer un même disque logique sur plusieurs disques physiques (dangereux), certaines où les blocs sont fragmentables, etc.

Structure du système de fichier sur un disque logique

Plusieurs disques logiques sur un disque physique



- **Boot bloc** utilisé au chargement du système.
- **Super Bloc** il contient toutes les informations générales sur le disque logique.
- **Inode list** Table des inodes.
- **blocs** les blocs de données chainés à la création du disque (mkfs).

Les blocs de données ne sont pas fragmentables sous Système V.

Leçon 4 : Principales commandes sur les fichiers Unix

Prérequis :

Chapitres précédents

Objectif du chapitre :

Connaître les commandes de manipulation des fichiers

Durée : 1h30

Eléments du Contenu :

Commandes de manipulation des répertoires

Commandes de manipulation des fichiers

LEÇON 4 : PRINCIPALES COMMANDES SUR LES FICHIERS UNIX

4-1- Principales Commandes sur les répertoires

pwd
print work directory
	Affiche le chemin d'accès du répertoire courant. Juste après connexion d'un utilisateur, la commande **pwd** lui précisera son répertoire d'accueil
	Exemple :
	`$ pwd`
	`/home/farhat`
	`$`

cd
change directory
	permet de changer le répertoire de travail
	Exemple :
	`$ cd ..`
	`Permet de remonter au répertoire père.`
	`$`

mkdir Make DIRectory	crée un nouveau répertoire **Exemple :** `$ mkdir tp0 tp1`
rmdir Remove Directory	supprime un répertoire, s'il est vide

4-2- Principales Commandes sur les Fichiers

On aura toujours besoins d'effectuer certaines tâches telles que lister le contenu du répertoire, copier, effacer ou afficher des fichiers. Nous présentons ci-dessous brièvement les commandes qui les réalisent :

ls List files	Lister et afficher les caractéristiques des fichiers contenus dans un répertoire **Options:** -**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
cat conCATenate	Une commande multi-usage qui permet d'afficher, de créer, de copier et de concaténer des fichiers
more	Permet d'afficher à l'écran un fichier page par page
cp Copy	permet de copier les fichiers **Options:** -**i** : demande confirmation -**p** : conservation de date et droits -**r** : recopie récursive d'un répertoire
mv Move	Déplace les fichiers et les répertoires
rm Remove	supprime la référence du fichier dans le répertoire **Options :** -**f** : force la commande sans s'occuper des droits -**i** : demande confirmation -**r** : destruction récursive
wc word count	permet le dénombrement de mots, lignes et caractères dans un fichier
ln Link	création d'une nouvelle référence pour un fichier préexistant **Option :** -**s** : création d'un lien symbolique
gzip/gunzip	compresse un fichier par défaut, la destination est la sortie standard qu'on redirige vers un fichier si on veut obtenir un fichier compressé **Options :**

	-**c** : résultat sans modification du fichier source -**f** : écrasement d'un fichier compressé préexistant -**v** : taux de compression -**d** : force une décompression -**9** : niveau maximum de compression
tar	archivage/désarchivage de fichiers **Options :** **c** : création de l'archive **x** : restauration de l'archive **t** : listage du contenu de l'archive **v** : mode verbeux **p** : préserve dates et droits à la restauration **f** : le nom du fichier spécifié sera celui de l'archive
touch	modifie les caractéristiques d'un fichier (option -**a** : m par défaut) crée un fichier vide s'il n'existe pas déjà **Options :** -**a** : dernier accès seulement -**m** : dernière modification seulement

Vous pouvez consulter le manuel man. Unix pour plus de détails

4-3 Copie de fichiers (cp)

La commande **cp** sert à faire la copie de fichiers. Elle peut copier un fichier sous un autre nom, copier un fichier d'un répertoire à un autre ou même copier un répertoire et tout son contenu dans un autre.

Exemple:

```
$ cp doc doc1
```

copie le fichier **doc** au fichier **doc1** Après la copie, les deux fichiers sont identiques.

Pour copier le fichier **doc** dans le sous-répertoire **projets**, il faudrait entrer :

```
$ cp doc projets
```

Pour copier le répertoire projets et tout son contenu dans le répertoire documents qui est au même niveau que le répertoire projets, on entrerait la commande cp avec l'option -r comme suit :

```
$ cp –r projets ../documents
```

Voici quelques exemples d'utilisation de la commande cp :

`cp f1 f2`	Copie le fichier f1 sous le nom f2.
`cp f1 rep`	Copie le fichier f1 dans le répertoire rep.
`cp * rep`	Copie tous les fichiers du répertoire en cours dans le répertoire rep.
`cp rep/f1 .`	Copie le fichier f1 du répertoire rep dans le répertoire en cours.
`cp -r /bin ~/bin`	Copie tous les fichier du répertoire /bin au répertoire bin de son compte.

4-4 Déplacement et suppression de fichiers (mv, rm)

La commande **mv** permet de renommer un fichier. Ainsi, la commande suivante :

```
$ mv devoir.c programme.c
```

Change le nom du fichier **devoir.c** à **programme.c**.

La commande **mv** permet aussi de déplacer un fichier d'un répertoire à un autre, en autant que ce soit à l'intérieur du même système de fichiers (la même partition du même disque). Par exemple, pour déplacer le fichier *image.jpeg* du répertoire en cours au répertoire documents, deux niveaux de répertoires plus haut, on entrerait :

```
$ mv image.jpeg ../../documents
```

Notez que, contrairement à la commande **cp**, la commande **mv** ne copie pas le fichier mais le déplace. Après le déplacement, le fichier ne se retrouve plus dans son répertoire initial.

Pour supprimer un fichier, il faut utiliser la commande **rm**. La commande rm supprime le fichier dont le nom est en paramètre. La commande suivante :

```
$ rm image.jpeg
```

supprimerait le fichier *image.jpeg* du répertoire ne cours.

Avec l'option **-r**, la commande **rm** supprimer un répertoire et tout son contenu. Par exemple, la commande suivante :

```
$ rm -r projets
```

supprimerait tout le répertoire projets, ainsi que tous les répertoires et tous les fichiers qu'il contient. Il faut utiliser cette option avec beaucoup de précaution. Souvenez-vous que, une fois supprimés, il est impossible de récupérer les fichiers car l'espace disque qui leur était allouée est déjà affectée à de nouveaux processus.

4-5 Les Liens

Les liens sont des fichiers spéciaux permettant d'associer plusieurs noms (liens) à un seul et même fichier. Ce dispositif permet d'avoir plusieurs instances d'un même fichier en plusieurs endroits de l'arborescence sans nécessiter de copie, ce qui permet notamment d'assurer un maximum de cohérence et d'économiser de l'espace disque. On distingue deux types de liens :

- Les **liens symboliques** représentant des pointeurs virtuels (raccourcis) vers des fichiers réels. En cas de suppression du lien symbolique le fichier pointé n'est pas supprimé. Les liens symboliques sont créées à l'aide de la commande ln -s selon la syntaxe suivante :

```
$ ln -s nom-du-fichier-reel nom-du-lien-symbolique
```

- Les **liens physiques** (aussi appelées liens durs ou en anglais hardlinks) représentent un nom alternatif pour un fichier. Ainsi, lorsqu'un fichier possède deux liens physiques, la suppression de l'un ou l'autre de ces liens n'entraîne pas la suppression du fichier. Plus exactement, tant qu'il subsiste au minimum un lien physique, le fichier n'est pas effacé. En contrepartie lorsque l'ensemble des liens physiques d'un même fichier est supprimé le fichier l'est aussi. Il faut noter toutefois qu'il n'est possible de créer des liens physiques qu'au sein d'un seul et même système de fichiers. Les liens physiques sont créées à l'aide de la commande ln (sans l'option -n) selon la syntaxe suivante :

```
$ ln nom-du-fichier-reel nom-du-lien-physique
```

4-6 Quelques utilitaires (diff, du, wc)

Vous pouvez utiliser la commande **diff** pour comparer deux fichiers. Cette commande relève les différences et les présente dans le fichier de sortie (à l'écran). Ainsi, la commande suivante :

```
$ diff prog1.c prog2.c
```

ferait une comparaison des fichiers *prog1.c* et *prog2.c* et afficherait les différences entre les deux, s'il en est. La commande **diff** n'affiche rien lorsque les deux fichiers sont identiques.

La commande **du** calcule l'espace disque utilisé par un répertoire et tout son contenu. Normalement, **du** affiche les nombres en termes de demi kilo-octets (512 octets), c'est pourquoi il vaut mieux toujours l'utiliser avec l'option -k pour obtenir les valeurs en kilo-octets (1 kilo-octet valant 1024 octets). Ainsi, la commande suivante :

```
$ du -k
23      ./devoir2
30 ./documents
1456    ./images
159      ./programmes-c
3194    .
```

calculerait l'espace disque occupé par chacun de ses sous-répertoires du répertoire en cours et enfin un total pour le répertoire en cours (total qui inclut ses propres fichiers, ainsi que ses sous-répertoires et leurs fichiers).

Avec l'option **-s**, la commande **du** ne donne que l'information totale, sans faire la répartition par sous-répertoire.

La commande **wc** calcule le nombre de lignes, de mots et de caractères que contient un fichier de texte. Ainsi, la commande suivante :

```
$ wc bonjour.c
     16      35     221 bonjour.c
```

indique que le fichier **bonjour.c** contient 16 lignes, 35 mots et 221 caractères. Pour n'avoir que l'un de ces trois nombres, il suffit de l'indiquer à l'aide d'une des options suivantes : **-l** pour le nombre de lignes, **-w** pour le nombre de mots et **-c** pour le nombre de caractères.

4-7- Recherche de fichiers (find):

La commande **find** parcourt les répertoires depuis le répertoire donné en paramètre à la recherche de fichiers qui satisfont certains critères et pose sur ces fichiers des actions. Elle a donc la forme suivante :

find *répertoire critères actions*

Par exemple, la commande qui suit explore le répertoire en cours (identifié par le point en premier paramètre) à la recherche de fichiers portant le nom prog.c (identifié par le mot-clef-name) et les énumère (action -print) :

```
$ find . -name prog.c -print
./devoir1/prog.c
./projet/prog.c
```

Critères de recherche

Dans le cas précédent, le critère utilisé est le nom du fichier. Mais **find** peut faire sa recherche en utilisant d'autres critères dont les suivants :

Critère	Explication
-atime n	Recherche les fichiers accédés depuis les *n* derniers jours.
-ctime n	Recherche les fichiers dont l'état (permissions, nom, groupe, etc.) a été modifié depuis les *n* derniers jours.
-mtime n	Recherche les fichiers modifiés depuis les *n* derniers jours.
-name n	Recherche les fichiers portant le nom *n*.
-size n	Recherche les fichiers dont la taille est de *n*. La valeur *n* est normalement exprimée en terme de blocs de 512 octets. Si *n* est suivi de c, la valeur est exprimée en terme d'octets.
-user u	Recherche tous les fichiers appartenant à l'utilisateur *u*.
-empty	Recherche tous les fichiers vides (de longueur zéro) et tous les répertoires vides (qui ne contiennent aucun fichier).

Il est possible de spécifier au critère -name des méta-caractères, mais il faut les mettre entre apostrophes pour éviter que le shell lui-même les interprète avant qu'ils ne parviennent à la commande **find** :;

```
$ find . -name '*.Z' -print
./logiciels/emacs-19.28.tar.Z
./logiciels/gcc-2.8.1.tar.Z
```

Les valeurs de -atime, -ctime, -mtime et -size peuvent être signées. Par exemple, la commande suivante recherche tous les fichiers modifiés depuis moins de 2 jours :

```
$ find . -mtime -2 -print
./java/com.class
./java/com.java
./langagec/fch.c
./langagec/liste
./langagec/liste.c
```

Les critères peuvent être combinés par l'un ou l'autre des opérateurs **-a** (ET) et **-o** (OU inclusif). L'exemple suivant cherche tous les fichiers dont la taille est supérieure à 100000 octets modifiés depuis moins de 30 jours :

```
$ find . -size +100000c -a -mtime -30 -print
./projet/gcc-2.8.1.tar.Z
./projet/apache-1.2.3.tar.Z
```

Actions exécutées

Nous avons déjà vu le fonctionnement de l'action -print, mais la commande **find** peut exécuter un certain nombre d'autres actions sur les fichiers sélectionnés :

Action	Explication
-exec c ;	Exécute la commande *c* en substituant le nom du fichier en traitement à {} dans l'énoncé de la commande.
-fprint f	Enregistre le nom du fichier avec chemin complet dans le fichier f.
-ls	Affiche le nom du fichier sous la forme de ls -dils.
-print	Affiche les noms des fichiers.

Par exemple, l'action -ls donne plus d'informations sur le fichier que l'action -print. Dans l'exemple qui suit, **find** cherche tous les fichiers vides (-empty) et les énumère (-ls) :

```
$ find . -empty -ls
800619   29 -rw-------  1 ISET etudiant  0 Dec 1
   1998 ./projet/journal
975298   26 -rw-------  1 ISET  etudiant 0 Mar 26
   1999 ./w3/cgi-bin/reponses
```

L'action -exec exécute une commande UNIX sur les fichiers sélectionnés. Il faut faire suivre le mot-clef -exec de la commande, puis d'un point-virgule (;) et **find** substitue le nom du fichier à une pare d'accolades ({}) dans la commande. Mais, comme le point-virgule final risque d'être interprété par le shell avant de parvenir à la commande **find**, il faut ou bien le faire précéder d'un oblique inversé (i.e. \;), ou bien le placer entre apostrophes (i.e. ';') :

La commande suivante cherche tous les fichiers vides depuis le répertoire travaux et exécute la commande ls -dils sur chacun des fichiers sélectionnés, ce qui équivaut à la commande précédente avec l'action -ls :

```
$ find travaux -empty -exec ls -dils {} \;
800619   29 -rw-------  1 ISET  etudiant      0 Dec  1
1998 ./projet/journal
975298   26 -rw-------  1 ISET  etudiant      0 Mar 26
1999 ./w3/cgi-bin/reponses
```

Bien sûr, la commande rm peut faire l'objet de l'action -exec, mais encore faut-il l'utiliser avec beaucoup de précaution. Une erreur est si vite arrivée et si lourde de conséquence. Pur cette raison, il est sage, dans un premier temps, de la remplacer par la commande ls et, après un contrôle, réexécuter la commande **find** avec la commande rm.

L'exemple qui suit combine des critères et utilise l'action -exec avec la commande rm. Elle efface tous les fichiers a.out et tous ceux portant l'extension .o qui n'ont pas été accédés depuis plus de 7 jours :

```
$ find . (-name a.out -o -name ´*.o´) -atime +7 -exec
rm {} \;
```

Pour plus d'information sur la commande **find**, lancez man find.

Leçon 5 : Droits d'accès aux fichiers Unix

Prérequis :

Chapitres précédents

Objectif du chapitre :

Connaître les techniques de protection des fichiers sous Unix

Durée : 1h30

Eléments du Contenu :

Droits d'accès sous Unix

Droits d'accès aux fichiers et répertoires

Modification des droits d'accès – Droits d'accès par défauts

Droits d'accès spéciaux

LEÇON 5 : DROITS D'ACCES AUX FICHIERS UNIX

5-1. Protection des fichiers

Comme tout système Multi-Utilisateurs, Unix possède des mécanismes permettant au propriétaire d'un fichier d'en protéger le contenu. Le propriétaire est l'utilisateur ayant créé le fichier. Pour permettre le partage de fichiers et faciliter le travail en équipe. Unix définit la notion de groupe d'utilisateurs, et tout utilisateur appartient à un groupe au moins.

Les droits d'accès (permissions) à un fichier sont définis par son propriétaire.

A chaque fichier est associé un ensemble d'indicateurs précisant les droits d'accès au fichier.

Pour chaque fichier il existe **trois types d'utilisateurs** :

Le propriétaire du fichier (**u**ser)

Les membres du groupe propriétaire du fichier (**g**roup)

Les autres utilisateurs du système (**o**thers)

Pour chaque fichier et par type d'utilisateur il existe trois modes d'accès

Autorisation d'écriture (**w**)

Autorisation de lecture (**r**)

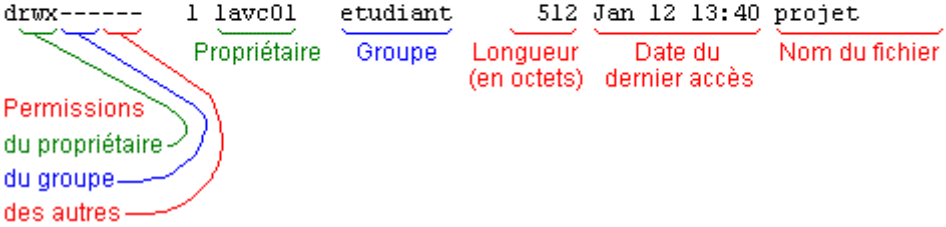
Autorisation d'exécution (**x**)

On obtient alors pour chaque fichier une combinaison de neufs bits (rwx rwx rwx) permettant de définir les permissions d'accès pour les fichiers

Pour visualiser les droits d'accès on utilise la commande ls -l. Le 1er caractère spécifie si le fichier est un répertoire (**d**), un fichier ordinaire (caractère **-**), un lien (l), un fichier en mode caractère(c), un fichier en mode bloc (b)… Les 9 caractères suivants identifient les droits d'accès (présence du droit si lettre r, w ou x ; absence de droit si caractère (-)

Exemple :

```
      $ ls -l
-rwxr-xr-x   2 lavc01   etudiant      456 Jan 11 12:59 convertir
-rw-------   3 lavc01   etudiant    29374 Jan 11 12:45 convertir.c
-rwxr-xr-x   1 lavc01   etudiant       78 Jan 11 12:58 convertir.sh
-rwx------   2 lavc01   etudiant    34196 Jan 15 17:38 devoir
-rw-------   3 lavc01   etudiant     1293 Jan 15 17:36 devoirl.c
drwx------   1 lavc01   etudiant      512 Jan 12 13:40 projet
```




```
- rwx r-x r--
```

Fichier ordinaire

Lecture, écriture et exécution permises pour le propriétaire

Lecture et exécution pour le groupe

Lecture pour les autres.

```
crw- rw- r--
```

Fichier spécial caractère

Lecture et écriture pour le propriétaire et pour le groupe

Lecture pour les autres

5-2 Modification des droits d'accès aux fichiers

5-2-1 Modification des droits d'accès :

La protection d'un fichier ne peut être modifiée que par le propriétaire à l'aide de la commande **chmod** (Change mode) Il existe deux modes d'utilisation de cette commande. La première (la plus ancienne) utilise la description des protections par un nombre octal.

Exemple

rwx rw– r–x est représenté par le nombre octal 765.

Une lettre équivaut à 1, un tiret à 0.

On a donc rwx rw– r–x = 111 110 101 = 765

Exemple

```
$ chmod 567 toto
```

Cette commande permet de modifier les droits d'accès au fichier toto de la façon suivante :

101 110 111

-r-x rw- rwx (cas très rare) autorisant un accès lecture et exécution au propriétaire, lecture et écriture au groupe et un accès sans restriction aux autres

Le deuxième mode d'utilisation de chmod, le mode symbolique, permet une description absolue ou relative des droits d'accès, comme suit :

`chmod [who] op [permission] fichier`

Où

who est une combinaison de lettre u (user = propriétaire), g (groupe), o (other = autre) ou a (all = tous) pour ugo,

op + permet d'ajouter un droit d'accès, – de supprimer un droit d'accès et = d'affecter un droit de manière absolue (tous les autres bits sont remis à zéro),

Permission r (read = lecture), w (write = écriture), x exécution).

Exemples

`Chmod u-w file`	Supprime le droit d'écriture au propriétaire.
`Chmod g+r file`	Ajoute le droit de lecture pour le groupe.

`Chmod ug= x file` Accès uniquement en exécution pour le propriétaire et le groupe, pas de modification pour les autres.

5-2-2 Droits d'accès à la création du fichier :

La protection d'un fichier, ainsi que le nom du propriétaire et le nom du groupe auquel vous appartenez, sont établis à sa création et ne peuvent être modifiés que par son propriétaire.

La commande **umask** permet de définir un masque de protection des fichiers (et répertoires) lors de leur création. Le masque est exprimé en base 8

Exemple :

```
$ umask 022
```

la valeur 022 est soustraite de la permission permanente (111 111 111)

111 111 111 Permission permanente

000 010 010 On enlève les bits du masque

111 101 101 = 755

umask 022 permet de créer des fichiers **répertoires** dont la protection est

rwx r-x r-x

Attention :

Pour les fichiers ordinaires, `umask 022` donnera une protection de type

rw- r- - r- -, car la possibilité d'exécution n'est pas autorisée par défaut sur les fichiers ordinaires.

5-2-3 Droits d'accès aux répertoires :

Dans le cas des répertoires, l'interprétation des droits est légèrement différente de celle concernant les fichiers. Les informations concernant un répertoire sont obtenues par la commande :

```
ls -ld rep
```

L'interprétation des protections pour les répertoires est la suivante:

r : autorise la lecture du contenu du répertoire comme dans le cas des fichiers ;

Permet de voir le liste des fichiers qui sont dans le répertoire (ls).

x : autorise l'accès au répertoire (a l'aide de la commande cd)

w : autorise la création, la suppression et le changement du nom d'un élément du répertoire. Cette permission est indépendante de l'accès aux fichiers dans le répertoire.

5-3. Les droits spéciaux :

Trois fois trois font neuf, le compte est bon me direz-vous ? vous auriez tort, car en réalité c'est trois fois quatre qui font douze. Il existe sous linux des droits spéciaux.

C'est ce qui permet par exemple que dans le répertoire /tmp auquel tout le monde a un accès plein droit, vous ne pouviez néanmoins supprimer les fichiers des autres utilisateurs.

5-3-1 Droits spéciaux sur les fichiers :

- **setuid**

Un fichier exécutable par son propriétaire, peut être setuid. C'est à dire qu'il s'exécute avec les droits de son propriétaire et non ceux de celui qui le lance. C'est d'ailleurs parce que "passwd" est setuid que vous pouvez modifier votre mot de passe avec "passwd" (ecrire dans /etc/passwd).

- **setgid**

De la même façon, un exécutable peut être setgid, et s'exécuter avec les droits du groupe auquel il appartient.

- **sticky bit**

Enfin, un exécutable peut être "sticky" : cela signifie qu'il reste en mémoire même après la fin de son exécution, pour pouvoir être relancé plus rapidement. **Attention :** seul root peut positionner le sticky bit.

5-3-2 Droits spéciaux sur les répertoires :

Il n'y a pas de **setuid** pour les répertoires, en revanche,

- **setgid**

Quand un répertoire est setgid, tous les fichiers créés dans ce répertoire appartiennent au même groupe que le répertoire. C'est utilisé par exemple quand plusieurs personnes travaillent sur un projet commun.

- **sticky bit**

Quand on positionne le sticky bit pour un répertoire, un utilisateur ne peut effacer que les fichiers qui lui appartiennent dans ce répertoire. C'est ce qui est utilisé pour le répertoire /tmp

5-3-3 chmod :

La logique est la même que précédemment, il suffit de rajouter une 4ème valeur octale, telle que setuid vaille 4, setgid vaille 2 et sticky bit vaille 1.

SETUID	SETGID	STICKY BIT
1	1	1

Si vous ne souhaitez pas jouer avec les droits spéciaux, faites un chmod avec 3 chiffres (par défaut les droits spéciaux conserveront leurs valeurs). Mais si vous souhaitiez par exemple que le contenu du répertoire /mnt/projet appartienne au groupe propriétaire du répertoire mais qu'en sus chacun ne puisse que ses fichiers, vous pouvez taper :

```
$ chmod 3777 /mnt/projet
```

Notez que c'est la colonne la plus à gauche qui code les droits spéciaux.

Leçon 6 : Gestion des Processus

Prérequis :

Système d'exploitation – chapitres précédents

Objectif du chapitre :

Connaître les caractéristiques des processus sous Unix

Durée : 1h30

Eléments du Contenu :

Notion théorique sur les processus

Etats d'un processus

Types de Processus – priorités – modes d'exécution

LEÇON 6 : GESTION DE PROCESSUS

6-1 Notion sur les processus

6-1-1 Définition

Un processus est un programme en cours d'exécution. Il a besoin de ressources matérielles : l'unité centrale, la mémoire centrale et l'accès à des périphériques d'entrées/sorties.

Les propriétés d'un processus appartiennent à ce que l'on appelle son environnement:

Parmi eux, le code, les données temporaires, les données permanentes, les fichiers associés...

6-1-2 Classification des processus

- Un processus est toujours créé par un autre processus appelé **processus parent.**
  - Ainsi tout processus a un processus parent sauf le tout premier. Ce tout premier processus est appelé **init**.
  - On appelle **processus fils** celui qui à été créé par un autre processus qui prend le nom de processus père.

- Deux types de processus existent :
  - *Les processus utilisateurs*

    Tous issus du *Shell* de connexion; Ils correspondent à chaque exécution d'un programme par l'utilisateur, le premier d'entre eux étant l'interpréteur de commandes à la connexion. Ces processus appartiennent à l'utilisateur et sont généralement attachés à un terminal.
  - *Les processus « démons » :*
    - Ces processus *daemon* assurent un service et sont souvent lancés au démarrage de la machine.
    - Les principaux services assurés par des processus *daemon* sont l'impression, les tâches périodiques, les communications, la comptabilité, le suivi de tâche.
    - Ces processus ne sont sous le contrôle d'aucun terminal et ont comme propriétaire l'administrateur du système ou un UID d'administration.

6-1-3 Caractéristiques

- Les caractéristiques statiques, c'est-à-dire ne variant pas au cours de sa vie, sont :
  - Un numéro unique : ***PID*** (*Process IDentifier*), est le numéro d'identification du processus. Il lui est attribué par le système à sa création

- Un **propriétaire** : **UID** déterminant les droits d'accès du processus aux ressources : ouverture de fichiers...
- Un **groupe** : **GID** déterminant les droits d'accès du groupe du processus
- Un processus **parent PPID** dont il hérite la plupart des caractéristiques,
- Un **terminal TTY** d'attache pour les entrées/sorties.

- Ses caractéristiques dynamiques sont :
  - **Priorité**, environnement d'exécution...
  - Quantité de ressources consommées (temps unité centrale utilisé...)
  - duré de traitement utilisé
  - référence au répertoire de travail courant
  - table de références de fichiers ouverts

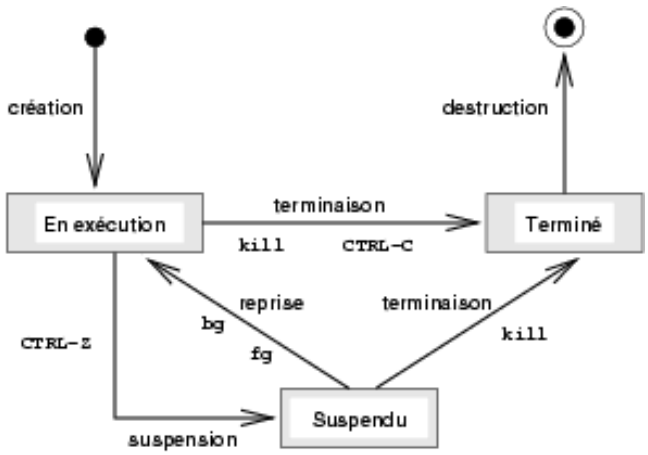
Pour chaque processus exécuté dans le système d'exploitation, sont stocké un certain nombre d'informations (environnement) dans une table appelée la table de processus, avec une entrée par processus en cours

6-2 Etats d'un processus

Le processeur exécute chaque processus *prêt* par petits intervalles. C'est comme si plusieurs processus avançaient en parallèle. Mais un processus n'est pas toujours prêt. Voici quelques uns de ses états possibles

CPU	c'est l'heureux élu. Il tourne
Runnable	prêt à être exécuté
Sleeping	en attente d'un événement, d'une ressource
Zombie	prêt à mourir
Stoppé	interdit de CPU
Swappé	complètement sur le swap

Pour un utilisateur, un processus peut se trouver dans trois états : **en exécution** (exécution de la commande), **suspendu** (CTRL-Z) ou **terminé**. Le schéma suivant récapitule les transitions permettant de passer d'un état à un autre.



6-3 Priorité d'un processus

Le processeur est attribué par tranches aux processus *prêts* en fonction de leur priorité. Cette priorité dépend de

- **classe**

  du processus (système, real-time, time-sharing, interactif, ...)

- **temps CPU**

  déjà utilisé

- **Nice**

  paramètre modifiable par l'utilisateur

Plus ***nice*** est petit, plus la priorité est grande. En version BSD elle varie de -19 à 19. En SYS V elle va de 0 à 39.

Un utilisateur ne peut qu'augmenter la valeur ***nice*** d'un processus lui appartenant, alors que le super-user a tous les pouvoirs

6-4 Modes d'exécution d'une commande:

Il existe cinq modes d'exécution d'une commande sous Unix:

Le mode interactif	La commande est lancée depuis l'interpréteur de commandes. Pendant l'exécution de la commande, l'utilisateur ne peut pas utiliser le terminal pour lancer une autre commande. Le contrôle n'est restitué à l'utilisateur qu'a la fin de l'exécution. On peut interrompre cette commande avec <ctrl + c> ou de la suspendre avec <ctrl + z>
Le Mode en arrière plan	Le lancement d'une commande en arrière plan (en ajoutant le caractère & à la fin de la commande) permet de rendre immédiatement le contrôle à l'utilisateur. Le mode arrière plan permet à l'utilisateur d'exécuter plusieurs tâches en même temps. Cela correspond à l'aspect multi-tâches du système Unix.
Le mode différé	L'exécution en différée d'une commande est réalise à l'aide de la commande **at** qui permet de déclencher l'exécution d'une commande à une date fixée.
Le mode batch	Permet de placer une commande dans une file d'attente. Le système exécutera toujours la commande placée en tête de la file d'attente. Ainsi toutes les commandes lancées par batch seront exécutées séquentiellement quelque soit l'utilisateur qui a mis la commande dans la file d'attente.
Le mode cyclique	Un mode qui permet d'exécuter une commande Unix de façon cyclique. Ce mode est réalisé a l'aide de la commande crontab .

Leçon 7 : Composition de processus

Prérequis :

Chapitres précédents

Objectif du chapitre :

Connaître les différentes manières de composition de processus et leurs utilités

Durée : 1h30

Eléments du Contenu :

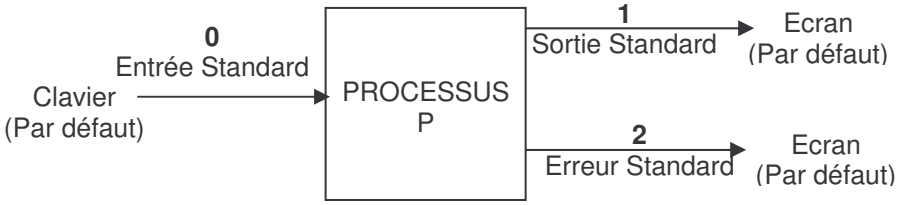
Composition de processus (séquentielle – parallèle)

Redirection des entrées / sorties des processus

LEÇON 7 : COMPOSITION DE PROCESSUS

7-1- La redirection des entrées sorties

Au lancement de chaque processus, l'interpréteur de commandes ouvre une entrée standard (par défaut le clavier), une sortie standard (par défaut l'écran) et une sortie d'erreur standard (par défaut l'écran)



Ces entrées-sorties standards peuvent être redirigées vers un fichier, un périphérique…

La redirection de la sortie standard consiste à renvoyer le texte qui Apparaît à l'écran vers un fichier. Aucune information n'apparaîtra à l'écran hormis celles qui transitent par la sortie d'erreur standard. Il est naturellement possible de rediriger toutes les entrées-sorties standard d'un processus, par conséquent le processus recherchera les informations dont il a besoins dans un fichier et non plus au clavier. Il écrira dans des fichiers ce qui devait apparaître à l'écran.

La re-direction des sorties peut être réalisé par effacement et création du fichier ou par ajout à la fin du fichier si ce dernier existe. Dans le cas contraire, un nouveau fichier sera créé. Dans le cas de la re-direction de l'entrée, il est évident que le fichier doit exister

Le caractère **<** suivi d'un nom de fichier indique la re-direction de l'entrée standard à partir de ce fichier

le caractère **>** suivi du nom de fichier indique la redirection de la sortie standard vers ce fichier

Si on double le caractère **>** la re-direction sera ajouté à la fin du fichier

Les caractères 2> suivis d'un nom de fichier indique la redirection de la sortie d'erreur standard vers un fichier

Remarque:

Si on souhaite rediriger la sortie standard et la sortie d'erreurs standards, la syntaxe sera:

Commande **>**fs **2>**erfs

Le caractère **&>** suivi du nom de fichier indique la redirection de la sortie standards et la sortie d'erreurs standards vers ce fichier.

Exemple:

```
$ ls>tmp
```

Liste dans le fichier tmp la liste des fichiers du répertoire courant

```
ls >> f.rep
```

Ajout a la fin du fichier f.rep la liste des fichiers du répertoire courant

```
ls /toto 2>tmp
```

Range dans le fichier tmp les messages d'erreurs et affiche sur l'écran la liste des fichiers

7-2- La composition de processus.

Un processus est un environnement d'exécution où, usuellement, un programme (commande, application, shell, ...) s'exécute. Cet environnement comprend des variables d'état et de choix. Du point de vue de l'utilisateur, plusieurs processus peuvent être **composés**. Par exemple, s'exécuter à un moment donné de façon concurrente, Ils peuvent aussi échanger des données.

L'Intérêt est de combiner des outils simples pour réaliser des fonctions complexes

Soient P1, P2 et P3 trois processus créés par le Shell pour exécuter des programmes C1, C2 et C3. il y a trois moyens pour composer ces processus

7-2-1 La composition séquentielle (;)

On la note **C1 ;C2 ;C3**

Les processus P1, P2 et P3, relatifs à C1, C2 et C3, seront enchaînés d'une façon indépendante et séquentielle.

Le déroulement du premier n'influe pas le second et ainsi de suite

Exemple:

```
$ pwd ; sleep 60 ; ls
/home/ig32
projet1 projet2 essai
```

Exécution de pwd puis de sleep 60 et ensuite ls

ls n'est exécutée qu'a la fin de sleep 60

7-2-2 La composition parallèle de processus indépendants (&)

Se note **C1&C2&C3**

Les trois processus P1, P2 et P3 sont créés simultanément et s'exécutent en parallèle.

Cependant, les processus P1 et P2 s'exécutent en arrière plan : ils sont détachés du clavier, et l'utilisateur ne peut pas leurs communiquer de données. Le processus P3 s'exécute lui en mode interactif. Si P1 et P2 sont privés du clavier, ils partagent avec P3 l'écran, leurs sorties et erreurs standards sont envoyées sur le même écran.

```
$ pwd & sleep 60 & ls
/home/ig32
projet1 projet2 essai
```

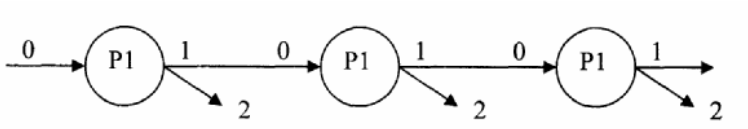
pwd sleep 60 et ls sont exécutés en parallèle.

Affichage immédiat du résultat de pwd et ls et sleep 60 continue en arrière plan

7-2-3 La composition parallèle de processus communicants (|)

Se note **C1|C2|C3**. Cette notation exprime qu'un **tube** relie la sortie de P1 avec l'entrée de P2, et qu'un autre tube relie la sortie de P2 avec l'entrée de P3. Les 3 processus sont lancés en parallèle, et se synchronisent automatiquement par leurs flots d'entrées-sorties

Un tube (pipe) est un flot de données qui permet de relier la sortie standard d'une commande à l'entrée standard d'une autre sans passer par un fichier temporaire



Exemple:

```
$ ls -l | more
```

Dans cet exemple, le résultat de la commande `ls - l` n'est pas affiché à l'écran, la sortie standard est redirigée vers l'entrée standard de la commande `more` qui quant à elle affiche son entrée standard page par page

```
$ who | wc -l
```

Compte le nombre d'utilisateurs connectés au système

```
$ ls | wc -w
```

Compte le nombre de fichiers dans le répertoire courant

Leçon 8 : Commandes de gestion de processus

Prérequis :

Chapitre précédents

Objectif du chapitre :

Connaître les principales commandes de gestion des processus

Durée : 1h30

Eléments du Contenu :

ps
top
kill
nice
Le Job Control (jobs)

LEÇON 8 : COMMANDES DE GESTION DE PROCESSUS

8-1-Commandes de gestion de processus

8-1-1 La commande PS:

L'aspect multi-tâches d'Unix est visible par la commande **ps**

La commande **`ps options`** permet d'obtenir des renseignements sur l'état des processus en cours. Par défaut, seuls les processus de la connexion en cours pour l'utilisateur ayant lancé la commande **ps** sont affichés.

L'option **–u nom_utilisateur**, permet de sélectionner les processus de cet utilisateur pour l'ensemble de ses connexions. Les autres options sont :

-e affiche des renseignements sur tous les processus en cours

-a affiche des renseignements sur tous les processus attachés à un terminal

-f génère pour chaque processus :

le nom de l'utilisateur (**UID**)

le numéro du processus (**PID**)

le numéro du processus père (**PPID**)

l'heure du lancement du processus (**STIME**)

le nom du terminal (**TTY**)

Temps d'exécution du processus (**TIME**)

Etat du processus : (**STAT**)

R en cours d'exécution

T arrêté (par <ctrl + z> par exemple)

P en attente de page

D en attente sur le disque

S en sommeil depuis moins de 20 s

I en sommeil depuis plus de 20 s

Z processus fils terminé

W « swape » sur le disque

COMMAND : La commande en cours d'exécution

Exemple

```
% ps
PID       TT STAT      TIME   COMMAND
24578 p7 Ss     0:00.33      -bash (bash-2.01)
24600 p7 S            0:00.52      xdvi.bin partie-03.dvi
24876 p7 R+     0:00.00      ps
```

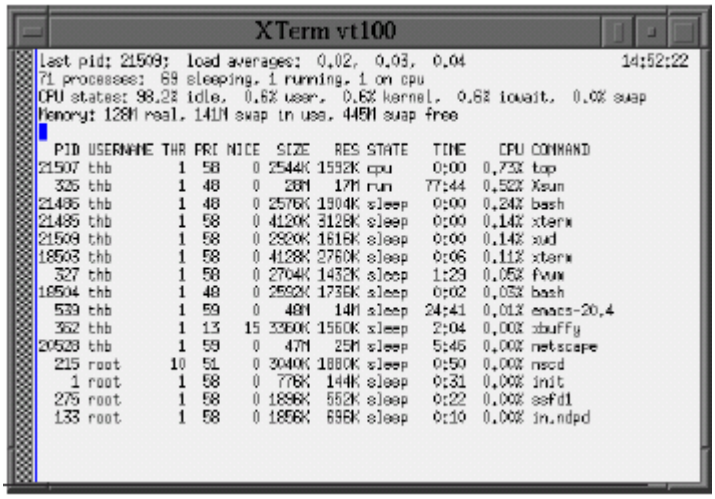
8-1-2 L'utilitaire top

Inconvénient de ps : c'est la liste des processus à un instant **t**.

On ne pourra jamais sous Unix avoir la liste des processus en cours : le temps de chercher les processus et de faire le rapport à l'écran, certains processus peuvent avoir disparu.

Amélioration de **ps** : la commande **top**

Son intérêt : elle affiche une liste des processus tous les n secondes



8-1-3 La commande kill

La commande **`kill -SIGNAL PID`** sert essentiellement à arrêter des processus en arrière plan (background). En effet, pour arrêter le ou les processus liés à la commande en cours d'exécution, il est beaucoup plus simple de taper <ctrl+c>

L'option `-SIGNAL` correspond au signal envoyé au processus. Un certain nombre d'évènement peut être signalé à un processus à l'aide d'un signal.

Ce mécanisme permet la communication inter-processus, notamment entre le système d'exploitation et le processus.

Les principaux signaux sont :

SIGHUP (SIGnal Hang UP : fin du shell)

SIGINT (SIGnal INTerrupt : interruption du programme)

SIGKILL (SIGnal KILL : tueur de processus

SIGSTOP (SIGnal STOP : stopper un processus)

SIGTERM (SIGnal TERMinate : terminaison)

8-1-4 La commande nice

Pour voir le **nice** d'un processus, on utilise la commande **top**. Pour modifier ce nice, utiliser top ou **renice**.

nice permet de lancer une commande passée en paramètre en abaissant sa priorité.

```
nice [-increment] commande_unix_normale [arguments]
```

Les processus Unix sont ordonnancés par le système suivant une priorité difficilement contrôlable par les utilisateurs. Ceux-ci ne peuvent généralement qu'abaisser cette priorité. Seul le super-utilisateur (root) peut l'élever.

La priorité est de 20 par défaut. Faites ps -efl pour afficher ce niveau. En dessous de 20, les processus sont plus prioritaires, en dessus ils le sont moins.

Le paramètre **incrément** est un nombre compris entre 1 et 19 qui positionne la priorité.

Si le niveau supérieur à 19, le système considère qu'il vaut 19.

Le super-utilisateur peut élever la priorité avec un argument increment introduit avec les caractères "--".

Sans préciser le niveau de priorité, celle ci est abaissée de 10.

La commande renice permet de modifie la priorité d'un processus.

8-1-5 La commande nohup

Lorsqu'un utilisateur se déconnecte, le shell envoie un signal SIGHUP à tous les descendants. Souvent (si le processus ne le capte pas ou ne l'ignore pas) cela amène à tuer un processus en background. La solution est d'utiliser la commande **nohup** qui va lancer le processus avec une option pour ignorer le **SIGHUP**.

Exemple:

```
$ nohup batch &
```

Notons toutefois que certains shells (dont les dérivés du C-shell) font un **nohup** automatique si on utilise le &.

8-2 Le job control

Le job est une ligne de commande shell. Il est composé d'un ou de plusieurs processus dans le cas d'utilisation de tube. Chaque job est numéroté de 1 à N par le shell.

Un job peut se trouver dans trois états :

Avant-plan (« foreground ») le job s'exécute et vous n'avez pas la main en shell

Arrière-plan (« background ») le job s'exécute et vous avez la main en shell

Suspendu (« suspended ») le job est en attente, il ne s'exécute pas

La commande **jobs** permet de lister les jobs en cours

Action	Etat initial du processus	Etat final du processus
$ commande		Processus en avant plan
$ Commande &		Processus en arrière plan
$ fg %num_job	Processus en arrière plan	Processus en avant plan
<ctrl> + z	Processus en avant plan	Processus stoppé
$ fg %num_job	Processus stoppé	Processus en avant plan
$ bg %num_job	Processus stoppé	Processus en arrière plan
$ kill -SIGSTOP %num_job	Processus en arrière plan	Processus stoppé

Leçon 9 : Les Filtres

Prérequis :

Chapitres précédents

Objectif du chapitre :

Introduire la notion de filtre et étude de quelques commandes

Durée : 1h30

Eléments du Contenu :

grep – cut – sort – tr - sed…

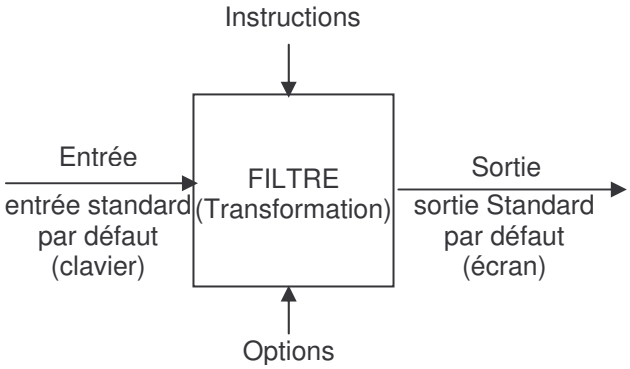
LEÇON 9 : LES FILTRES

9-1- Définition

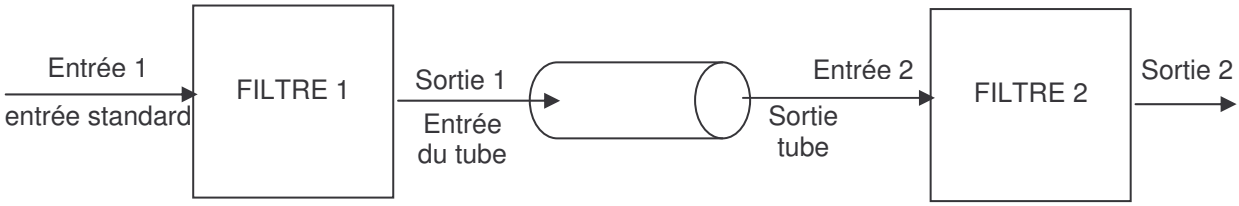
Un **filtre** est un programme (commande ou encore utilitaire) Unix qui lit une entrée, qui sera considéré comme un flot composé de *lignes de caractères* et réalise une ou plusieurs **transformations** sur chaque ligne pour produire un résultat.

Pour un filtre l'entrée provient d'un fichier qui est, par défaut, l'entrée standard (le clavier). De même le résultat est écrit sur un fichier qui est, par défaut, la sortie standard (l'écran).

Dans le cas général, les transformations dépendent des instructions et sont contrôlées par des options.



Parmi les commandes qui jouent le rôle de filtres citons : **grep** (dont **egrep** est une variante), **tail** (affichage d'une partie d'un fichier), **cat**, **sort**, **wc**, **cut** , **sed** (éditeur de texte non interactif) et **awk** qui supporte un véritable langage de programmation, proche du C, pour la manipulation des fichiers texte.



Les filtres sont particulièrement intéressants lorsqu'ils sont enchaînés à l'aide de **tubes**

Exemples de Filtres

cut	Sélectionne des caractères selon leur position dans la ligne
grep	Recherche des occurrences d'expressions dans les fichiers
sort	Trie, fusionne un ou plusieurs fichiers
tr	Substitution ou suppression de caractères sélectionnés
tee	rediriger une commande tout en la transmettant à un tube

9-2- Filtre grep

Recherche des occurrences d'expressions dans les fichiers dont on fournit la liste.

Syntaxe

```
grep [options] expression  [fichiers ...]
```

Options	Signification
-c	Nombre de ligne trouvées (sans les afficher).
-i	Ne fait pas la différence entre majuscule et minuscule.
-n	Affiche le numero de la ligne.
-l	Affiche le nom du fichier contenant la ligne (et pas la ligne).
-v	Affiche toutes les lignes qui ne contiennent pas le mot en question.

La syntaxe de **expression** peut cependant être beaucoup plus complexe et représenter un ensemble de chaînes de caractères, c'est ce que l'on appelle une **expression régulière**.

Nous allons utiliser la puissance des expressions régulières pour coder toutes ces possibilités avec des **caractères spéciaux**.

Les caractères spéciaux et leur signification

Caractère	Signification
[...]	Plage de caractères permis.
[^...]	Plage de caractères interdits.
^	Début de ligne.
.	Un caractère quelconque, y compris un espace.
*	Caractère de répétition, agit sur le caractère placé avant l'étoile. Accepte également l'absence du caractère placé devant lui.
$	Fin de ligne.
\{...\}	Répétition.
\{Nombre\}	Répétition de *Nombre* exactement.
\{Nombre,\}	Répétition de *Nombre* au minimum.
\{Nombre1 Nombre2\}	Répétition de *Nombre1* à *Nombre2*.

Exemples

1. Afficher toutes les lignes du fichier prog.f contenant read
   ```
   $ grep read prog.f
   ```

2. Afficher avec la numérotation de toutes les lignes du fichier prog.f contenant read
   ```
   $ grep -n read prog.f
   ```
3. Affichage de toutes les lignes du fichier prog.f contenant la chaîne de caractère « call sub » en majuscules ou en minuscules
   ```
   $ grep -i 'call sub' prog.f
   ```
4. Affichage de la ligne (ou des lignes) contenant un caractère numérique dans tous les fichiers du répertoire courant
   ```
   $ grep '[0-9] *
   ```
5. Recherche de tous les fichiers contenant la chaîne de caractères unix et affichage de leurs noms
   ```
   $ grep -l 'unix' *
   ```

9-3- Filtre sort

Tris et ordonnancement de fichiers selon des critères (des clefs) précisés en option en s'appuyant sur la notion de champs

Si plusieurs noms de fichiers sont donnés, le résultat est une combinaisons triée de ces fichiers. Le tri peut être restreint à une partie de chaque ligne, un champ délimité par des séparateurs.

c1 et c2 sont des numéros de champs : l'option +c1 permet d'exclure du tri les champs de 1 à c1 ;

l'option –c2 permet d'exclure du tri les champs qui suivent le champ c2.

Si c2 n'est pas précisé, le tri se fera du champ c1+1 jusqu'à la fin de la ligne.

Syntaxe

```
Sort [-options] [+c1 [-c2] ] ..[fichier..]
```

Options :

-c	test si l 'entrée est déjà triée
-m	fusion des fichiers d'entrée qui doivent être triés
-d	tri seulement sur l'ordre alphabétique
-i	caractères de contrôle ignorés
-n	tri numérique
-r	ordre de tri inversé
-tx	choix du séparateur de champs x au lieu de la valeur par défaut (espace ou tabulation)
-o fileres	résultat du tri dans le fichier fileres

Exemples :

1. Tri du fichier essai ; le résultat du tri sera dans le fichier essai.tri
   ```
   $ sort -o essai essai.tri
   ```
2. Tri du fichier /etc/passwd sur le numéro de l'utilisateur (UID)
   ```
   $ sort -t : +2n -3 -o /etc/passwd.tri /etc/passwd
   ```

9-4-Filtre cut

Filtre qui permet de couper en plusieurs champs des lignes d'un fichier ou de l'entrée standard.

Syntaxe :

```
$ cut -cliste [f1……]
$ cut -fliste [-dx][f1……]
```

Options :

-dx	choix du caractère x comme séparateur de champs
-cliste	indique la position des caractères (position en colonne)
-fliste	indique la position des champs

Exemples

- Affichage des portions de lignes se trouvant en vingtième position (caractères) pour le fichier f1
  ```
  $ cut -20c f1
  ```
- Affichage du nom et du numéro (UID) de chaque utilisateur (champs 1 et 3 avec séparateur : du fichier /etc/passwd)
  ```
  $ cut -d : -f1,3 /etc/passwd
  ```

9-5- Filtre tr

Translate character

Filtre qui permet la substitution ou suppression de caractères sélectionnés.

Syntaxe

```
tr [options] chaine1 chaine2
```

Remplace toutes les occurrences de TOUS les caractères de chaine1 par le caractère de chaine2, de même rang, dans le flot d'entrée

Options

-d : suppression des caractères sélectionnés

-s : “aaaaa” dans chaine1 devient “a” dans chaine2

Exemples

- Convertit tous les caractères majuscules de fichier1 en caractères minuscules dans fichier2
  ```
  tr '[A-Z]' '[a-z]' < fichier1 > fichier2
  ```
- Convertit les caractères <TAB> (code ascii octal 011) de fichier1 en caractères espace (" ") dans fichier2
  ```
  tr '\011' ' ' < fichier1 > fichier2
  ```
- supprime toutes les minuscules non accentuées
  ```
  $ echo $ligne | tr -d a-z
  ```
- supprime les espaces multiples entre les mots
  ```
  $ echo $ligne | tr -s ' '
  ```

9-6 L'utilitaire sed

Il s'agit d'un utilitaire (sed = "Stream EDitor") qui sélectionne les lignes d'un fichier texte (ou d'un flot provenant d'un pipe) vérifiant une expression régulière et qui leur applique un traitement ou un remplacement.

L’utilitaire **sed** est un éditeur de texte en ligne. Contrairement aux autres éditeurs comme **ed** et **vi** qui agissent sur un fichier en le modifiant

directement, sed agit sur un fichier d'entrée pour produire une version modifiée dans un fichier de sortie distincte du fichier d'entrée.

Syntaxe

```
sed [-n] [-e script] [-f fichier-commandes] fichier-source
```

- L'option -n empêche la sortie à l'écran du résultat (souvent associé à l'option p)
- Le fichier source est traité ligne par ligne conformément à la liste des commandes (-e) ou au fichier de commandes (-f)

9-6-1 Commande de substitution

La commande s permet d'effectuer des substitutions suivant la syntaxe:

```
sed [adresse]s/expr-régulière/remplacement/options
```

Attention! contrairement à ce que l'on pourrait attendre, cette commande laisse passer toutes les lignes et ne sélectionne pas celles qui ont satisfait l'expression régulière et donc subi la substitution. Pour sélectionner, voir la commande de destruction.

Options

- Sans précision, la commande ne s'applique qu'à la 1ère occurrence de chaque ligne.
- 0...9 : indique que la substitution ne s'applique qu'à la nième occurrence
- g : effectue les modifications sur toutes les occurrences trouvées.

Exemple

- le fichier fich.moi est parcouru, à chaque occurrence de "moi", ce mot est remplacé par "toi" et le nouveau fichier est sauvegardé sous le nom fich.toi

```
$ sed s/moi/toi/g fich.moi > fich.toi
```

- Le slash / étant très utilisé au niveau du shell comme séparateur de niveau de répertoire, il est possible d'utiliser à la place tout autre caractère comme #

```
sed s#/home#/rep_perso#g /etc/passwd > /tmp/passwd.new
```

9-6-2 Destruction ou sélection

Cette option permet de filtrer les lignes qui satisfont une expression régulière. Ces lignes ne sont pas détruites dans le fichier d'origine, mais ne sont pas transmise en sortie.

comment modifier alors le fichier à traiter ?

cp fichier copie
sed /.../d copie
Par exemple, pour détruire toutes les lignes vide d'un fichier :

```
$sed /^$/d
```

9-6-3 Ajout, insertion et modification

Pour utiliser ces commandes, il est nécessaire de les saisir sur plusieurs lignes

```
$ sed [adresse] commande\
```

La commande peut être :

a pour ajout ;
i pour insertion ;
c pour modification.

9-6-4 Exemples

- Substitue toutes les occurences de Chien ou de chien par Médor
  ```
  sed 's/[Cc]hien/Médor/g' fich_entree > fich_sortie
  ```

- Détruit toutes les fins de lignes qui commencent par le caractère #
  ```
  sed 's/#.*$//' fich_entree > fich_sortie
  ```

- Détruit toutes les lignes commençant par # ou par !
  ```
  sed '/^[#!]/d' fich_entree > fich_sortie
  ```

- Détruit toutes les lignes ne contenant pas (!) un chiffre
  ```
  sed '/[0-9]/!d' fich_entree > fich_sortie
  ```

- Affiche fichier sauf lignes No x à No y ($ pour spécifier dernière ligne)
  ```
  sed 'x,yd' fichier | more
  ```

- Affiche lignes se trouvant entre lignes BEGIN et END (inclues)
  ```
  sed -n '/^BEGIN/,/^END/p' fichier
  ```

- Insère contenu du fichier adresse après chaque ligne contenant iset, et sauve sur fichier référence toutes les lignes contenant info
  ```
  sed -e '/iset/r adresse' -e '/info/w reference'
  entree > sortie
  ```

- Exécution des commandes sed se trouvant dans fich_commandes_sed
  ```
  sed -f fich_commandes_sed ...
  ```

Leçon 10 : Programmation Shell

Prérequis :

Chapitres précédents

Objectif du chapitre :

Introduire les notions de base de la programmation Shell sous Unix

Durée : 1h30

Eléments du Contenu :

Script Shell – déclaration de variables – Les variables spéciales - Le jeu d'instruction du Shell

LEÇON 10 : PROGRAMMATION SHELL

10-1- Introduction :

Le shell est plus qu'un interpréteur de commandes : c'est aussi un langage de programmation.

Unix offre la possibilité d'enregistrer dans des fichiers des suites de commandes que l'on peut exécuter par la suite. Ces fichiers s'appellent des scripts.

Sous Unix existe plusieurs shell, ce qui nous conduit en fait à plusieurs langages de programmation.

Dans tout ce qui suit, nous allons utiliser le shell BASH

Comme tout langage de programmation conventionnel, le shell comporte des instructions et des variables.

Les noms des variables sont des chaînes de caractères, et leurs contenus sont également des chaînes de caractères.

10-1-1 Les variables du Shell (langage)

Les noms de variables sont des chaînes de caractères et leurs contenus sont également des chaînes de caractères.

L'affectation (assignation) d'une valeur à une variable se fait par un nom.

La référence à cette variable se fait par un nom précédé du caractère $.

Exemple :

```
mavariable = bonjour  (affectation)
echo $mavariable
```

10-1-2- Jeu d'instructions du shell

Le jeu d'instructions du shell comporte :

Toutes les commandes Unix

Invocation de programmes exécutables (ou de scripts) avec passage de paramètres.

Des instructions d'affectation de variables

Des instructions conditionnelles et itératives (if, while, for, ...)

Des instructions d'entrées / sorties

Remarques :

Les mécanismes de tubes et de redirections sont utilisables dans un script.

Le shell est un langage interprété

Il est possible d'invoquer un script écrit dans un shell différent du shell interactif

Commencer le script par la ligne `# !/bin/xxx`, chemin d'accès du shell qui doit interpréter le script

10-2- Programmation de base en shell

10-2-1 Premier script

Créer avec vi un fichier
Ajouter des commandes
Enregistrer
Ajouter le droit x
Le script peut être exécuté comme une commande.

exemple :

```
$ vi script0
echo bonjour
```

10-2-2- Passage de paramètres :

On peut passer des paramètres (arguments) a des scripts lors de leurs invocations. Pour ce faire on désigne par 1..2..3.....9 les variables qui permettent de designer respectivement le 1er, 2ème,.....,9ème paramètre associé a l'invocation du script.

exemple :

```
script la
# !/bin/bash
ls -la
$ chmod u+x la
$./la
```

On peut modifier « la » de la façon suivante :

```
# script la
# !/bin/bash
echo "contenu du répertoire
$1"
ls -la $1

exécution
$ la /home/ig32
```

Le nombre de paramètres passés en arguments à un script n'est pas limité à 9. toutefois seuls les neufs variables 1... 9 permettent de designer ces paramètres dans le script.

La commande shift permet de contourner ce problème. Après exécution de shift le $i^{ème}$ paramètre est désigné par $i-1

exemple:

```
script echopara
echo $1 $2 $3
p1 = $1
shift
echo echo $1 $2 $3
echo $p1
```

```
exécution
$ echopara un deux trois
un deux trois
deux trois
un
```

10-2-3- Variables spéciales

En plus des variables 1...9
Le shell prédéfinit des variables facilitant la programmation comme:

0 contient le nom sous lequel le script est invoqué
le nombre de paramètres passés en arguments
? le code de retour de la dernière commande exécutée
$ contient le numéro (PID) du shell (décimal)

exemple :

```
script echopara2
echo $0
a été appelé avec $# paramètres
echo qui sont : $*
Execution
$ echopara2 a b c d
echopara2 a ètè appelé avec 4 paramètres
qui sont : a b c d
```

10-2-4 instructions de lecture/écriture

Ces instructions permettent d'instaurer un dialogue interactif sous forme de questions/réponses.
La question est posée par l'ordre **echo** et la réponse est obtenue par l'ordre read a partir du clavier.
read variable1 variable2... variablen

exemple:

```
script affiche
echo "nom du fichier à afficher"
read fichier
cat $fichier
```

Leçon 11 : Programmation Shell (structures de contrôle)

Prérequis :

Chapitres précédents

Objectif du chapitre :

Connaître et utiliser les structures de contrôle du Shell et écrire des scripts pouvant effectuer des traitements complexes

Durée : 1h30

Eléments du Contenu :

Les Structures répétitives – les structures conditionnelles – arithmétique entière sur les variables

LEÇON 11 : PROGRAMMATION SHELL (STRUCTURES DE CONTROLE)

11-1- les structures de contrôle:

- instructions conditionnelles
- itérations

11-2- Les instructions conditionnelles:

il existe trois outils

- l'instruction **if**
- commande **test** qui la complète
- instruction **case**

11-2-1- Instruction if

Elle présente trois variantes :

1-

```
if commande1
then commandes2
fi
```

Les commandes2 sont exécutés si la commande1 renvoie un code retour nul ($? =0)

exemple:

1-

```
if grep -i "ig32" /etc/passwd
then echo ig32 est connu du système
fi
```

2-

```
if commande
then commandes1
else commandes2
fi
```

3-

```
if commande1
then commandes1
elif commande2
then commandes2
elif commande3
then commandes3
...
...
else
commandes0
fi
```

11-2-2 La commande test

Indispensable complément de if

Elle permet de :

Reconnaître les caractéristiques de fichiers et des répertoires

Comparer les chaînes de caractères

Comparer algébriquement les nombres

Deux syntaxes différentes:

1-

```
test expression
```

ou

2-

```
[  expression  ]
```

```
if [<espace>expression<espace>]
then commandes
fi
```

Les expressions les plus utilisées sont:

-d nom vrai si le rep nom existe

-f nom vrai si le fichier nom existe

-s nom vrai si le fichier nom existe et non vide

-r nom

-w nom

-x nom

-z chaîne vrai si la chaîne de caractère est vide

-n chaîne vrai si la chaîne existe et non vide

c1=c2 vrai si c1 et c2 identiques

c1 !=c2 vrai si c1 et c2 différents

n1 –eq n2 vrai si les entiers n1 et n2 sont égaux

(-ne, -lt, -le, -gt, -ge)

11-2-3 instruction case:

```
case chaîne in
motif1) commandes1;;
motif2) commandes2;;
...
...
motifn) commandesn;;
esac
```

Le shell recherche parmi les différents chaînes de caractères motif1, motif2,... motifn proposés, la première qui correspond à chaîne et exécute les commandes correspondantes.

Exemple:

```
case $# in
0) echo $0 sans arguments;;
1) echo $0 possède un argument;;
2) echo $0 a deux arguments;;
*)echo $0 a plus que deux arguments;;
esac
```

11-3- Les itérations

La présence des instructions itératives dans le shell en fait un langage de programmation complet et puissant. Le shell dispose de trois structures itératives: **for**, **while** et **until**.

11-3-1 Itération bornée: la boucle for

Trois formes de syntaxes sont possibles:

Forme1

```
for variable in chaine1 chaine2 chaine3...chainen
do
commandes
done
```

les valeurs de variable sont les chaînes chaine1 chaine2 chaine3 ...chainen

Forme2

```
For variable
do
commandes
done
```

chaîne prend ses valeurs dans la liste des paramètres du script

Forme3

```
for variable in *
do
commandes
done
```

chaîne prend ses valeurs dans la liste des fichiers du répertoire

Remarque:

Pour chacune des trois formes, les commandes placées entre **do** et **done** sont exécutées pour chaque valeur prise par la variable du shell « variable ».

Ce qui change c'est l'endroit où « variable » prend ses valeurs.

exemple:

```
script lsd
echo "liste des répertoires sous pwd"
for i in *
do
          if [ -d $i ]
          then
          echo $i " est un répertoire"
          fi
   done
```

11-3-2 Itérations non bornées : while et until

```
while commande1
do commandes2
done
```

```
until commandes1
do commandes2
done
```

Les commandes commandes2 sont exécutées tant que (while) ou jusqu'à ce que (until) la commande commande1 retourne un code nul (la condition est vraie)

Exemple 1:

Script qui liste les paramètres qui lui sont passés en arguments jusqu'a ce qu'il rencontre le paramètre fin.

```
#script finwhile
while [ $1 != fin ];do
echo $1
shift
done
$ finwhile 1 2 3 4 fin 5 6 7 8
1
2
3
4
```

$

```
# avec until
until [ $1 = fin ]; do
echo $1
shift
done
$ finuntil 1 2 3 fin 4 5 6
1
2
3
$
```

11-4 Arithmétique entière sur les variables.

Le shell ne permet pas de définir des variables numériques, et pourtant les opérateurs –eq, -ne –lt –le –ge existent.

Ce n'est pas une incohérence, cars les opérateurs précédents peuvent s'appliquer en cas où le contenu de ses variables sont des nombres entiers.

La commande `expr` permet en outre d'évaluer une expression arithmétique sur de telles variables. Le résultat de cette évaluation est une chaîne de caractères dirigée sur la sortie standard.
Les opérations permises sont les quatre opérations arithmétiques + - * / et %

Exemple

```
read a
read b
c = 'expr $a + $b'
echo $c
```

Références bibliographiques

- Linux Initiation et Utilisation (1er et 2ème cycle. IUT. Ecoles d'Ingénieurs

Jean-Paul Armspach / Pierre Colin / Frédérique Ostré-waerzeggers
Edition Dunod 2002

- Unix et les systèmes d'exploitation

Cours et exercices corrigés
Michel Divay
Edition Dunod 2002

- Administration Système Unix

Thierry Besançon – Philippe weil
Cours de formation permanente – Université Pierre et Marie Curie – Paris 6

- Cours Unix

Jaque Menu – Université de Genève
http://cui.unige.ch/cours/techInternet/