Enseignement d'Informatique : II21

Titre : Introduction du système UNIX

Objectif : Introduire le système d'exploitation UNIX en étudiant sa structure et en manipulant son language de commande SHELL.

Contenu :

Bibliographie :

Maurice J. Bach "Conception du système UNIX" Masson paris 1989
Jean-Marie Rifflet "La programmation sous UNIX" Ediscience International 1998

Remerciements :

Je remercie Gérard Berthelot , avec lequel j'ai eu le plaisir d'assurer l'enseignement d'UNIX à l'Institut d'Informatique d'Entreprise (IIE) du CNAM, de m'avoir permis de reprendre son support de cours et de pouvoir le modifier.

Date de dernière modification : 17 septembre 2002

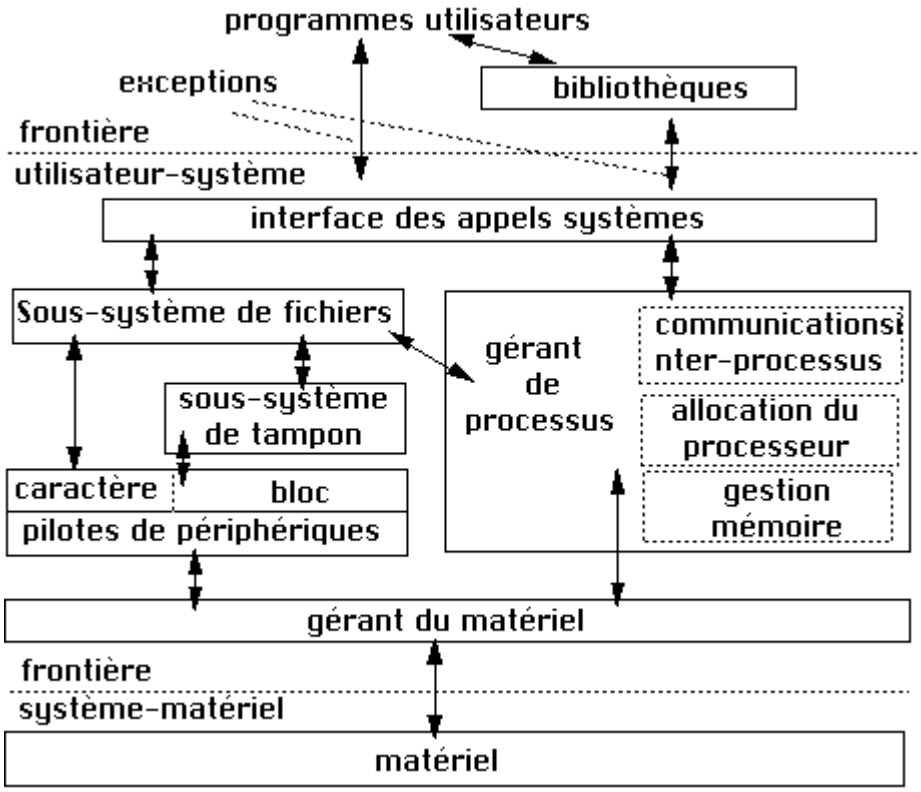
1 Présentation d'UNIX

1.1 Historique

UNIX créé au Laboratoire BELL, USA, en 1969 .
Destiné à la gestion d'un mini-ordinateur pour une petite équipe de programmeurs.
Intéresse rapidement de nombreuses universités puis des constructeurs.
Deux principales familles de systèmes UNIX (1983): **Berkeley(BSD)** et **System V** de Bell.
De nombreux efforts de normalisation : norme System V , **POSIX**(1988), OSF
De nombreuses versions d'UNIX sont donc apparues :
ULTRIX (BSD) puis OSF sur DIGITAL, IRIX(System V) sur Silicon Graphics,
LINUX(POSIX - 1991) sur PC, et bien d'autres.

1.2 Les Concepts Fondamentaux

Système multi-utilisateurs et multi-tâches, indépendant de toute architecture matérielle/constructeur.
Permet la répartition des ressources (mémoire, processeurs, espace disque, imprimantes, programmes et utilitaires) entre les utilisateurs et les tâches.
Unix est constitué d'un gestionnaire de fichiers et d'un gestionnaire de processus.
Chaque utilisateur peut exécuter plusieurs programmes simultanément.
Fournit des primitives pour construire des applications complexes à partir d'autres plus simples, avec une interface graphique puissante- X-Windows (MIT-1985) , MOTIF.
Il est possible de rediriger les entrées et sorties des processus.
Un mécanisme de communication par tubes permet de synchroniser des processus et de leur faire échanger des informations.
Un système UNIX est administré par un super-utilisateur ("superuser").



2. Le Système de Gestion de Fichiers

2.1 Les Différents Types de Fichiers

* les fichiers ordinaires ou fichiers réguliers : programmes, données, ... Ces fichiers sont destinés à recevoir des données, des utilisateurs ou du système. Leur unité d'accès pour la lecture ou pour l'écriture est l'octet. Un utilisateur peut considérer les fichiers de données comme des tableaux de caractères munis d'un index de lecture/écriture ;
* les catalogues ou répertoires ("directory"). Les répertoires sont des fichiers de données particuliers utilisés par le système pour organiser les ensembles de fichiers qu'il doit gérer en une structure "arborescente". Le contenu d'un répertoire est une suite de couples formés d'un nom de fichier et d'un numéro (index par lequel le système identifie le descripteur d'un fichier dans la table des i-noeuds) ;
* les fichiers spéciaux : utilisés pour spécifier les périphériques. Ils se trouvent dans le catalogue /dev ("device") et sont référencés par un utilisateur comme des fichiers ordinaires.

2.2 Arborescences, Liens Physiques

On a un système de fichiers arborescent dont les noeuds sont les noms des catalogues non vides et les feuilles sont les noms des catalogues vides et des fichiers non catalogues. Chaque utilisateur dispose de sa propre sous-arborescence.
Il est possible de créer des noms "synonymes" qui permettent de désigner le même fichier ordinaire avec deux noms placés dans des répertoires différents. Le synonyme d'un nom existant est appelé un lien physique.
Il existe également des liens symboliques se référant à des répertoires ou systèmes de fichiers.

2.3 Références Absolues et Références Relatives

Les fichiers sont désignés par des "références" qui doivent désigner de façon unique un seul fichier dans une arborescence.
La racine du système de fichiers est unique et est référencée par /. La référence absolue d'un fichier est $/nom_1/nom_2/.../nom_n$.
La désignation par les références absolues est souvent fastidieuse . De plus, son utilisation dans les programmes fige l'arborescence.
Le catalogue de travail ("working directory") désigne le catalogue à partir duquel il faut à chaque instant rechercher d'autres fichiers par référence relative.
Tout répertoire contient deux références particulières, **.** et **..**, qui désignent respectivement le répertoire lui-même et son prédécesseur dans l'arborescence .
A chaque utilisateur est associé un catalogue privé ("home directory") qui est le catalogue de travail au login.

2.4 Les Principaux Répertoires

Les principaux répertoires ont les mêmes noms sur tous les systèmes UNIX :

* /bin et /usr/bin contiennent les programmes associés aux commandes UNIX non internes aux différents langages de commandes ;

* /dev contient les fichiers spéciaux ;

* /etc contient les fichiers système ;

* /tmp contient les fichiers temporaires.

2.5 Les Droits d'Accès et leur Gestion

Tout utilisateur possède un numéro d'utilisateur et le numéro du groupe auquel il appartient. On distingue trois types d'utilisateurs potentiels :

- le propriétaire du fichier ("user", ou en abrégé u);
- les utilisateurs appartenant au même groupe ("group", en abrégé g);
- les autres utilisateurs ("other", en abrégé o).

Trois types d'opérations sur les fichiers sont possibles :

- la lecture ("read", en abrégé r);
- l'écriture ("write", en abrégé w);
- l'exécution ("execute", en abrégé x).

Il y a donc 9 combinaisons possibles utilisateur-opération. C'est pourquoi les protections sont codées sur 9 bits.
3 autres bits donnent le type d'un fichier :
-: ordinaire
d : repertoire
b : spécial bloc
c : spécial caractère
p : pipe
s : socket
l : lien physique/symbolique

3. Interpréteurs de Commandes

Un interpréteur de commande (ou "shell" en terminologie UNIX) est un processus lancé par le système UNIX lorsqu'un utilisateur se connecte. Il est attaché au terminal qui sert à établir la connexion. Il est chargé de recueillir les commandes émises par l'utilisateur et si possible de les exécuter.

3.1 Connexion/Déconnexion

Une fois le terminal allumé , le message suivant apparaît à l'écran :
login :. taper son nom d'utilisateur suivi de la touche **<CR>**. En réponse à la question suivante, i.e. **password :**, le mot de passe de l'utilisateur doit être composé. Le mot de passe est évidemment tapé en aveugle, pour que personne ne puisse le lire. Ensuite, un certain nombre d'informations sont affichées, suivies d'un caractère d'invite ("prompt") qui peut être le caractère $, le nom de la machine, ou autre chose, dépendant du site sur lequel on se trouve. Le "prompt" est envoyé par l'interpréteur du langage de commandes shell pour informer l'utilisateur qu'il est en attente de lecture de commandes.
Le système différencie les minuscules et les majuscules.
La connexion permet la création d'une session de travail sous le SHELL.

3.2 Syntaxe et Exécution d'une Commande SHELL

Une commande : nom arg1 arg2 ...
Les arguments qui sont des options ou des paramètres. Les différents éléments de la commande doivent être séparés par des blancs ou des caractères de tabulation.
Pour l'exécution d'une commande deux cas sont possibles :

- soit le nom de l'action est le nom d'un fichier contenant un programme exécutable (commande externe),
- soit l'action doit être exécutée par l'interpréteur de commandes lui-même(commandes interne).

* Pour les commandes internes l'interpréteur exécute la commande puis affiche le résultat et attend la commande suivante.
* Les commandes externes, qui correspondent à des fichiers exécutables sont utilisées pour avoir les interpréteurs les plus simples possibles et pour permettre de réutiliser des commandes développées pour un autre interpréteur

3.3 Caractères Spéciaux

Certains caractères jouent un rôle particulier :

- ***** représente toute chaîne de caractères ;
- **?** désigne un caractère quelconque (**.** dans l'éditeur **ed**) ;
- **[...]** désigne un caractère quelconque de l'ensemble spécifié dans **...**. Par exemple, [abc1] représente un caractère appartenant à {a, b, c, 1}. Des abréviations permettent de ne pas décrire explicitement tout l'ensemble : [0-9], [a-z], [A-Z] ;
- > et < représentent des redirections d'entrées/sorties standard ;
- | désigne un tube ;
- **;** permet de construire une séquence ;
- <**CR**> passe à la ligne de commandes suivante.

Si l'on désire utiliser ces caractères en tant que caractères normaux, il faut les faire précéder d'un caractère \ qui perd lui-même sa particularité lorsqu'il est doublé. Lorsqu'une référence de fichier commence par un **.**, ce premier caractère doit être spécifié explicitement car cette occurrence n'est pas couverte par la convention *****.

3.4 Aide en Ligne (man)

man [*section*] *titre*
affiche page à page le chapitre correspondant au *titre* donné dans le manuel standard d'Unix. Certains titres se trouvent dans plusieurs *sections*.

- Les commandes externes sont en section 1,
- les appels système en section 2 et
- les fonctions des bibliothèques standard en section 3.

Man *com*
Permet d'obtenir de la documentation sur la commande com.

3.5 Quelques Commandes Utiles

* **who** affiche la liste des utilisateurs connectés.
* **more *nom_fichier*** affiche page à page le contenu de *nom_fichier*.
* **find** permet de trouver des fichiers dans l'arborescence.
  - Critères de recherche : nom, permissions d'accès, type, nombre de références, propriétaire, groupe du propriétaire, taille, numéro de i-noeud, dates de création, de modification.
  - Ces critères peuvent être combinés à l'aide d'opérateurs logiques "et" et "ou".
  - Lorsqu'un fichiers a été trouvé , la commande find peut lui appliquer certaines opérations.
    - find / -name titi -print
    cherche récursivement à partir de la racine tous les fichiers de nom titi et affiche leur référence absolue au fur et à mesure.

Exemple :

```
Pelvoux:/home/cemif-sc/mallem/cours/ii21/poly> who
Root       console      sep   2 11:00    (:0)
Root       pts/3        sep   2 11:01    (:0.0)
Mallem     pts/4        sep  16 09:59    (gsc18.cemif.univ-evry.fr)
Dcasa      pts/5        sep  16 10:13    (info01.iup.univ-evry.fr)
```

Quelques Commandes Utiles (suite)

* **grep *chaîne_caractères liste_noms_fichiers*** affiche, pour chaque fichier référencé dans *liste_noms_fichiers*, les lignes contenant *chaîne_caractères*.
* **passwd** permet de créer ou de changer le mot de passe.
* **tty** affiche le nom du terminal utilisé.
* **wc [-lwc] [*liste_noms_fichiers*]** ("word count") compte et affiche, pour tous les fichiers référencés dans *liste_noms_fichiers* le nombre de lignes, mots et caractères. Si aucun nom de fichier n'est précisé, la commande utilise l'entrée standard. L'option **-l** ("line") permet de n'afficher que le décompte des lignes, l'option **-w** ("word") celui des mots et l'option **-c** ("character") celui des caractères.

Exemple :

```
Pelvoux:/home/cemif-sc2/mallem/cours/ii21/poly> who | grep mallem
Mallem     pts/4        sep  16 09:59    (gsc18.cemif.univ-evry.fr)
```

3.6 Les Commandes de Manipulation de Fichiers

* **pwd** ("print working directory") affiche la référence absolue du catalogue de travail.
* **cd [*nom_catalogue*]** ("change directory") change de catalogue de travail. Si aucun paramètre n'est fourni, le catalogue de travail devient le catalogue privé de l'utilisateur, sinon le catalogue de travail devient celui dont la référence est *nom_catalogue*.
* **ls** affiche la liste des fichiers dans le catalogue de travail. **ls *liste_références*** affiche, pour tout élément de *liste_références* qui référence un fichier ordinaire, son nom, et pour tout élément de *liste_références* qui référence un catalogue, son contenu. **ls -l** affiche aussi les protections, la taille, la date de dernière modification. **ls –a** permet de voir les fichiers "silencieux" dont le nom commence par un **.** (**.cshrc** et **.login** qui sont exécutés au login)
* **cp [-i] *ancien nouveau*** ("copy") effectue une copie physique d'un fichier dans un autre : création d'un nouvel i-noeud, d'une nouvelle entrée dans un catalogue et recopie effective du contenu du fichier *ancien* dans *nouveau*

Exemple :

```
Pelvoux:/home/cemif-sc2/mallem/cours/ii21/poly> cd ../..
Pelvoux:/home/cemif-sc2/mallem/cours/ii21/poly> pwd
/home/cemif-sc2/mallem/cours/
Pelvoux:/home/cemif-sc2/mallem/cours/ii21/poly> ls
ii21/  ii31/
```

Les Commandes de Manipulation de Fichiers (suite)

ln [-s] *fichier_existant fichier_nouveau* ("link") crée un lien de nom *fichier_nouveau* sur *fichier_existant* . (-s lien symbolique)

Exemple :

```
Pelvoux:> ls -l ii21/poly
-rwxr-xr-x  1 mallem      24576 Sep 16 11:48 a.out*
drwxr-xr-x  2 mallem        512 Sep 16 11:48 doc/
-rw-------  1 mallem        245 Sep 16 11:46 essai.c
Pelvoux:> ln -s doc doc2
Pelvoux:> ls -l
-rwxr-xr-x  1 mallem      24576 Sep 16 11:48 a.out*
drwxr-xr-x  2 mallem        512 Sep 16 11:48 doc/
lrwxrwxrwx  1 mallem          3 Sep 16 13:44 doc2 -> doc/
-rw-------  1 mallem        245 Sep 16 11:46 essai.c
```

Les Commandes de Manipulation de Fichiers (suite)

* **mv [-i] *ancien_nom nouveau_nom*** ("move") renomme le fichier *ancien_nom* en *nouveau_nom*. L'option **-i** demande confirmation si *nouveau_nom* est le nom d'un fichier existant déjà (sinon il est écrasé).
* **rm [-ir] *liste_noms_fichiers*** ("remove") supprime, pour chaque nom de fichier dans *liste_noms_fichiers*, la référence du fichier. Le compteur de références de l'i-noeud correspondant est décrémenté de 1. L'i-noeud et le fichier physique ne sont détruits que si le compteur de références devient nul. L'option **-i** demande confirmation avant chaque suppression. L'option **-r**, permet de supprimer récursivement le contenu du catalogue puis le catalogue lui-même.
* **mkdir *liste_noms_catalogues*** ("make directory") crée les catalogues vides référencés dans *liste_noms_catalogues*.
    - **rmdir *liste_noms_catalogues*** ("remove directory") pour chaque catalogue vide référencé dans l*iste_noms_catalogues*, supprime le catalogue.
* **chown [-R] *nom_utilisateur*[*.nom_groupe*] *liste_noms_fichiers*** ("change owner") pour chaque fichier référencé dans *liste_noms_fichiers*, le propriétaire du fichier devient *nom_utilisateur*. L'option **-R** permet d'appliquer récursivement la commande aux catalogues de la liste. Chown est réservée à l'administrateur.
* **chmod [-R] *nouvelles_protections liste_noms_fichiers*** ("change mode") pour chaque fichier référencé dans *liste_noms_fichiers*, le mode de protection devient *nouvelles_protections*.
    - Protections définies par rapport à : user, group, other

- Protections autorisent (ou non) : read, write, execute

Les protections sont affichées par la commande ls -l sous la forme d'une chîne de 9 caractères : rwxrwxrwx qui indiquent les droit du propriétaire, du groupe et des autres, en lecture, ecriture et exécution. Si un caractère est remplacé par un - l'accès correspondant est refusé à la catégorie correspondante.

- ***Nouvelles_protections*** peut être exprimé de deux manières :
  - 750 correspond en octal à rwxr-x--- (1 = autorisé, 0 = interdit)
  - ug+w rajoute les autorisations d'écriture au propriétaire et au groupe
- ***Type de fichiers :*** Il existe 3 autres bits qui représentent le type de fichier

X X X r w x r w x r w x
Type
User(Owner)
Group
Other

Type :
- : ordinaire
d : directory
p : pipe
s : socket
c : caractere
b : bloc
l : lien (symbolique/physique)

– ***Autres attributs :*** Il existe 3 autres autres combinaisons qui positionnent des bits spéciaux :
04000 : positionne le **suid bit** (le fichier s'exécute avec les droits du propriétaire)
*X X X r w **s** r w x r w x (le 'x' du propriétaire est transformé en 's')*

02000 : positionne le **sgid bit** (le fichier s'exécute avec les droits du groupe)
*X X X r w x r w **s** r w x (le 'x' du groupe est transformé en 's')*

01000 : positionne le **sticky bit** (le fichier est maintenu en mémoire après son exécution, ex : compilateur …)
*X X X r w **t** r w x r w x (le 'x' du propriétaire est transformé en 't')*

3.7 La Commande df

L'espace physique sur le(s) disque(s) physique(s) est divisé en disques logiques référencés dans le catalogue des fichiers spéciaux. Chacun des disques logiques possède sa propre table d'index et sa propre racine. Les disques logiques peuvent être montés. Ils deviennent alors accessibles.
df [-i] [liste_noms_disques_logiques] ("display filesystem") affiche la taille des disques logiques montés, la place utilisée et disponible, le pourcentage de la capacité d'un disque logique déjà utilisé. L'option **-i** permet d'afficher le nombre d'i-noeuds utilisés et disponibles. On peut noter que la somme de la taille disque utilisée et de la taille disque disponible est plus petite que la taille totale du disque logique. Ceci est dû au fait que le système réserve une partie du disque logique (en général 10%) pour ne pas avoir de problème lors des allocations. Seuls un super-utilisateur et le système lui-même peuvent se servir de cette marge.

Exemple :

```
Pelvoux:> df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda1              3943796   1590276   2353520  40% /
/dev/sda1              4373060   2443896   1795908  58% /home/iup
193.105.81.240:/home/etudiant
                       7147296   5279408   1867888  74% /home/etudiant
/dev/sdb1              4373292     21636   4129500   1% /home/cemif-sc
```

3.8 La Commande ps

ps [[-]aef] [-t*term*] affiche des informations sur les processus sous forme de tableau. **-aef** (System V) permet de voir les informations complètes sur tous les processus en cours.

- colonne **UID** ("user identification") : numéro de l'utilisateur concerné.
- colonne **PID** ("process identification") : numéro du processus concerné.
- colonne **TT** ("terminal") : nom du terminal .
- colonne **TIME :**temps CPU utilisé par le processus
- colonne **COMMAND :** nom complet de la commande
- colonne **STAT** ("status") : état du processusdécrit par 4 caractères
    - première lettre (indique l'état de fonctionnement du processus) :
    - **R** ("runable") processus en cours d'exécution;
    - **T** processus arrêté;
    - **S** ("sleep") processus en sommeil pour moins d'environ 20 secondes;
    - **I** ("idle") processus en sommeil pour plus d'environ 20 secondes;
    - **Z** ("zombie") processus ayant terminé qui attend que son père exécute une attente ("wait");
    - La seconde lettre (indique si un processus est swappé) :
    - La troisième lettre (indications sur la priorité du processus) :
    - La quatrième lettre indique des traitements particuliers mém. virtuelle.

Exemple :

```
Pelvoux:/home/cemif-sc2/mallem/cours/ii21/poly> ps
     UID   PID  PPID  C    STIME TTY      TIME CMD
  Mallem 21119 20883  9 14:32:13 pts/4    0:02 ls -F -R /
  Mallem 20883 20881  0 13:32:32 pts/4    0:01 -csh
```

[mallem@aramis ~]$ **ps -aef | grep mallem**
root 30601 30598 0 09:54 pts/1 00:00:00 login -- mallem
mallem 30799 30601 0 09:54 pts/1 00:00:00 -tcsh
root 10854 10852 0 09:59 pts/4 00:00:00 login -- mallem
mallem 11073 10854 0 09:59 pts/4 00:00:00 -tcsh
mallem 11616 11073 0 09:59 pts/4 00:00:00 ps -aef
mallem 11617 11073 0 09:59 pts/4 00:00:00 grep mallem
[mallem@aramis ~]$

3.9 Arrêt d'un Processus : la Commande kill

Les signaux sont un mécanisme permettant à un utilisateur ou au système d'avertir un processus qu'un événement particulier concernant ce processus s'est produit.
kill [*-num*] *num_proc1 num_proc2 ...* envoie le "signal de terminaison" aux processus de numéros *num_proc1*, *num_proc2*, ... L'option ***-num*** permet d'envoyer un autre signal dont la nature est spécifiée par le numéro est *num* plutôt que le "signal de terminaison". On utilise souvent le signal 9 ("signal tueur" ou SIGKILL) qui permet de tuer les processus en toutes circonstances.
Sur un grand nombre de systèmes UNIX, le même effet est obtenu, pour les processus qui ne s'exécutent pas en tâche de fond, en frappant CTRL-C (respectivement CTRL-\) qui provoque l'envoi d'un "signal de terminaison" (respectivement d'un "signal tueur"). Mais certains processus sont récalcitrants. Il n'y a alors pas d'autre possibilité que de se reloger pour utiliser la commande kill.
Pour des raisons évidentes, l'utilisation de la commande kill sur les processus d'un autre utilisateur est réservée à l'administrateur du système.

Exemple :

```
Pelvoux:> kill -9 21119
Pelvoux:> ps
     UID   PID  PPID  C    STIME TTY      TIME CMD
  Mallem 20883 20881  0 13:32:32 pts/4    0:01 -csh
```

3.10 Les Redirections

Les entrées et sorties des processus se font a priori sur des canaux spécifiques. Il en existe trois :
* l'entrée standard qui est par defaut associée au clavier, ;
* la sortie standard qui est par défaut associée à l'écran du terminal;
* la sortie erreur standard qui est aussi par défaut associée à l'écran du terminal, sur laquelle le processus inscrit ses messages d'erreur.

On peut vouloir modifier les entrées/sorties d'un processus par exemple parce que les entrées sont contenues dans un fichier ou les sorties sont beaucoup trop longues pour qu'on puisse les voir à l'écran, et on préfère alors les mettre dans un fichier. Pour cela, on effectue une redirection du canal adéquat.

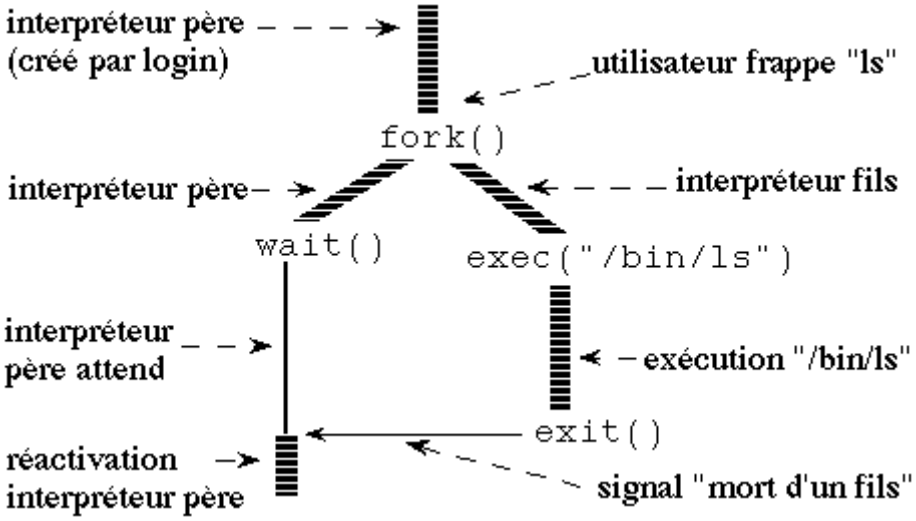
Redirections (suite)

Redirection de l'entrée standard
Comm* < *nomfich la commande *comm* prend ses entrées dans le fichier *nomfich*.
Redirection de la sortie standard :
Comm* > *nomfich redirige les sorties de la commande *comm* sur le fichier *nomfich*. Si ce fichier existe déjà, il est écrasé, sinon, il est créé;
Comm* >> *nomfich redirige les sorties de la commande *comm* sur le fichier *nomfich*. Si ce fichier existe déjà, les sorties de la commande sont inscrites à la suite du contenu du fichier (donc sans écrasement), sinon, le fichier est créé.
Redirection de la sortie erreur standard :
Comm* 2> *nomfich redirige les erreurs générées lors de l'exécution de la commande *comm* sur le fichier *nomfich*. Si ce fichier existe déjà, il est écrasé, sinon, il est créé.
Exemple :

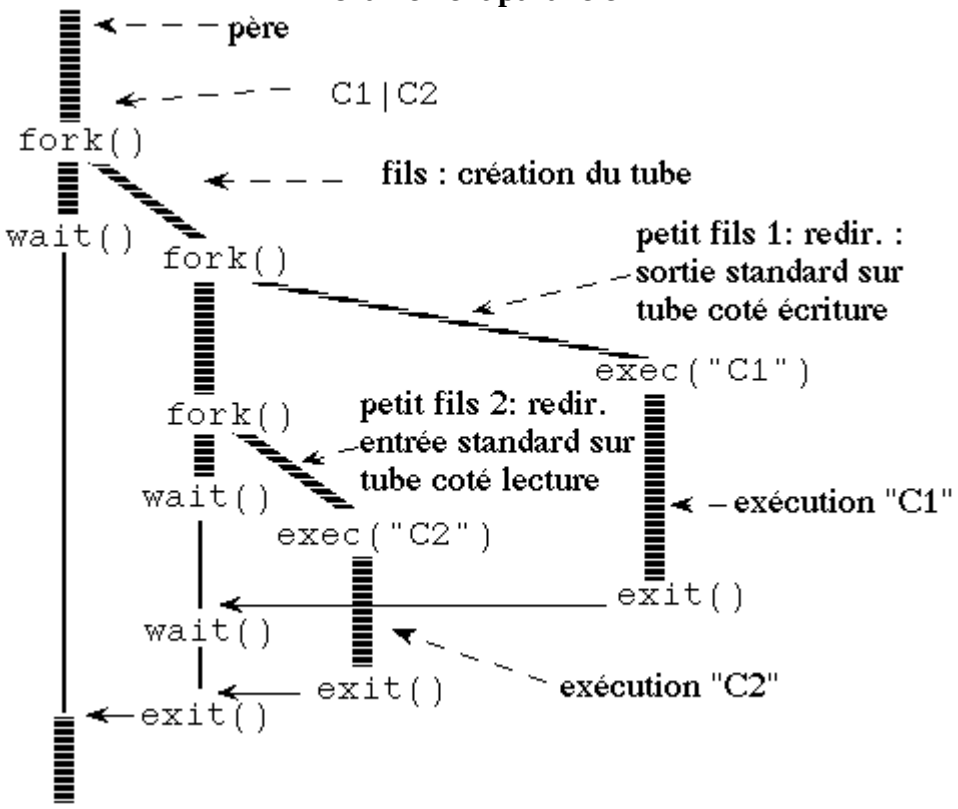
```
Pelvoux:>grep main essai.c >filtre
Pelvoux:>more filtre
Main(argc,argv)
Pelvoux:>
```

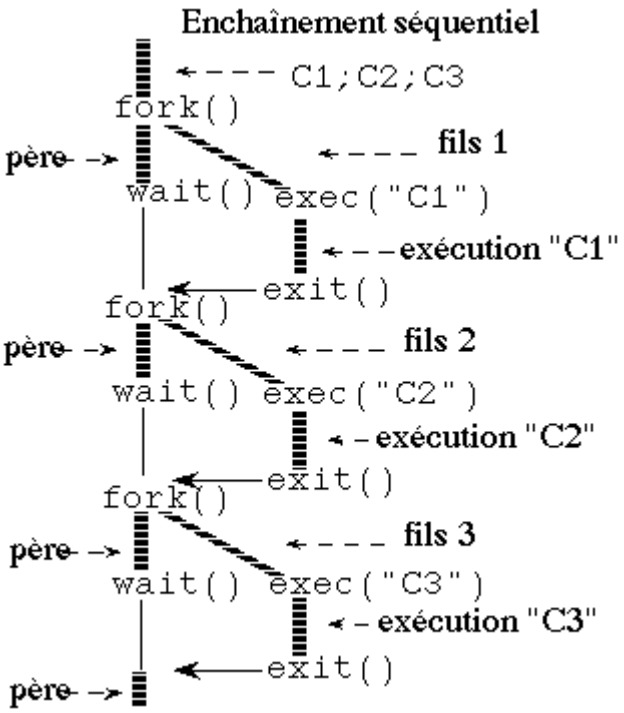
3.11 Exécution de Processus

Création d'un processus : **fork** Substitution de processus : **exec** Attente de fin de processus : **wait** Terminaison de processus : **exit**



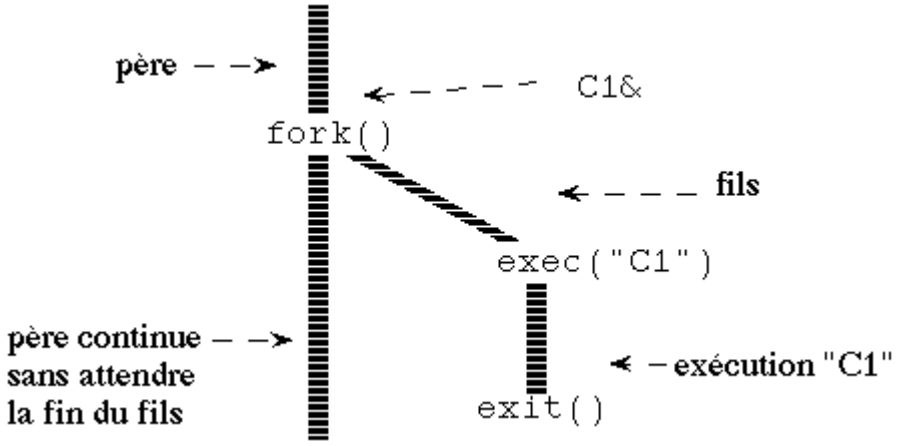
Enchaînement parallèle





3.12 Tâche de Fond

comm& lance la commande *comm* en tâche de fond, l'interpréteur n'attend pas la fin de l'exécution de la commande et permet de relancer immédiatement une seconde commande. Le système affiche le numéro du processus créé.
Exécution d'une commande en tâche de fond :



4 Langage de Commande (Shell)

4.1 Shell, Login-Shell

Le Shell est à la fois un langage de commandes et l'interpréteur de ce langage.
Il assure l'interface externe entre les utilisateurs et le système, mais ne fait pas partie du noyau.
Principaux langages de commandes sont disponibles sur les systèmes UNIX :

- le Bourne shell (**sh**),
- le C-shell (**csh**) basé sur **sh**,
- **tcsh**, basé sur **csh**, qui est du domaine public,
- **ksh** (korn shell) ,
- **bash** (Linux).

4.2 Commandes Externes et Commandes Internes

L'interpreteur du shell a la possibilité, soit d'exécuter lui même la commande demandée par l'utilisateur (*commande interne*), soit de lancer l'exécution de programmes existant par ailleurs pour ce faire (*commandes externes*).
À chacune des commandes externes correspond un fichier exécutable ayant comme nom le nom de la commande. Ce fichier se trouve souvent dans le répertoires */bin*
L'exécution d'une commande externe entraîne la création d'un nouveau processus exécutant le programme correspondant à la commande.
Une commande externe peut être appelée à partir de n'importe lequel des langages de commandes disponibles, mais aussi à partir d'un programme utilisateur.
Chacun des langages de commandes dispose de ses propres commandes qui sont les commandes internes.

4.3 Le Login et l'Environnement Shell

Lorsqu'un utilisateur se logue, un processus shell est exécuté. De plus, certaines commandes, soit communes, soit propres à chacun, sont effectuées :

- **.profile**,
- **.login**
- **.cshrc**

Ce type de commande permet l'initialisation de variables shell, soit pour leur donner une valeur différente de celle par défaut, soit pour en définir de nouvelles.
L'ensemble de ces variables constitue l'environnement shell.

4.4 Les variables

Le nom d'une variable shell est une chaîne de caractères contenant des lettres, des chiffres ou le caractère _ et commençant toujours par une lettre.
La valeur d'une variable est une chaîne de caractères quelconque.

Une affectation de la valeur *val* à la variable *var* s'effectue par :

var = val.

Ou

let var=<expression>
ou <expression> peut être :
- statut de retour d'une commande,
- résultat d'une commande entre ``,
- expression arithmétique,
- opération sur les chaînes de caractères.

La valeur d'une variable *var* est obtenue en utilisant *$var*. Si la variable *var* n'a pas été définie, son contenu est la chaîne de caractères vide.

Il est également possible d'exporter une variable(la rendre globale) :
export var
La commande export sans argument donne la liste des variables exportées.

On distingue plusieurs catégories de variables :

- Les variables définies par l'utilisateur,
- Les variables d'environnement,
- Les variables réservés au SHELL(cf. 4.6).

4.4.1 Les variables d'environnement existant toujours sont :

* **PS1** : premier prompt ;
* **PS2** : second prompt. Celui-ci est utilisé en particulier pour continuer une commande commencée mais pas terminée ;
* **HOME** : référence absolue du catalogue privé ;
* **PATH** : liste de chemins dans lesquels les commandes appelées vont être cherchées ;
- **TERM** : type du terminal utilisé;
- **USER** : Nom de l'utilisateur connecté;
- **SHELL** : Nom du SHELL pris par défaut
- **ENV** : Liste des variables d'environnement.

Ces variables sont consultables par la commande **env** et sont modifiables par **setenv**
Il est possible de définir de nouvelles variables d'environnement :
Ex:
```
set MON_PROJET=~/src/projet/c++
cd $MON_PROJET
```

4.4.2 Lecture et affichage de variables

read permet la lecture d'une ou plusieurs valeurs à affecter à des variables :

read var1 var2 …
Les valeurs numériques respectives des différentes variables sont séparées par des espaces. L'affectation des variables est effectuée apres la validation(<CR>).

echo permet l'affichage un texte(son argument) sur l'écran, ce texte peut contenir des variables. Dans ce cas la variable est précédée de $. La validation de **echo** provoque un saut de ligne.

Echo "texte1 $var texte2"
echo -n "texte" évite un retour à la ligne
echo -e "salut \t à tous " prend en compte les caractères non visualisables

4.4.3 Interprétation par le SHELL des chaînes de caractères

Des délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères :
* '**...**' : **...** est pris tel quel, c'est-à-dire que si cette chaîne contient des appels à des variables, aucune substitution n'est effectuée ;
* "**...**" : les variables contenues dans **...** sont substituées ;
* **`op`** : évalue la commande shell **op**.

4.4.4 Les Procédures et leurs Paramètres

Une procédure (ou *script shell*) est une suite de commandes shell. Elle peut accepter des paramètres: **Commande paramètre$_1$ ... paramètre$_n$**. Ces variables reservées au SHELL sont accessibles en lecture seule.
Le premier paramètre est référencé par **$1**, le second par **$2**, ..., le neuvième par **$9**.

- **$1**, **$2**, ..., **$9** : paramètres (de position) de la commande (si plus de 9 utiliser $*);
- **$0** : nom de la commande(script) appelée ;
- **$*** : liste des paramètres à partir de $1;
- **$#** : nombre de paramètres passé au script;
- **$$** : numéro du processus shell correspondant à la commande ;
- **$?** : code de retour de la dernière commande exécutée(vaut 0 si la commande s'est bien passé).

Les variables utilisées dans une commande(script) sont locales à celle-ci.

La commande interne **shift** effectue un décalage de pas +1 dans les paramètres de position ($1 prend la valeur de $2) et la variable $# est décrémentée de 1 ainsi de suite …

La commande **set** var1,var2… ou **set** `commande`permet d'initialiser $1,$2,…

4.5 Structures de Contrôle

Affectation

Set <chaîne de caractères> : la <chaîne de caractères> devient la nouvelle liste de paramètres.
Read <liste de variables> : les variables prennent les valeurs fournies par l'entrée standard.

Conditionnelles :

if <liste de commandes 1>
then <liste de commandes 2>
else <liste de commandes 3>
fi

if <liste de commandes 1>
then <liste de commandes 2>
elif <liste de commandes 3>
then<liste de commandes 4>
else <liste de commandes 5>
fi
.
case <chaîne de caractères> **in**
<motif1> **)** <liste de commandes 1> **;;**
...
<motif n> **)** < liste de commandes n> **;;**
***)** <liste de commandes pour les autres cas> **;;**
esac

Structures de Contrôle (suite)

Répétitions

for <variable> **in** <liste de chaînes de caractères>
do <liste de commandes>
done
.
for <variable> **in** $*
do <liste de commandes>
done
.
while <liste de commandes 1>
do <liste de commandes 2>
done
.
until <liste de commandes 1>
do <liste de commandes 2>
done

Structures de Contrôle (suite)

Test <condition>

L'instruction test permet d'effectuer de nombreux tests aussi bien sur des fichiers, des valeurs numériques ou des chaînes de caractères. Le résultat sous forme booléenne est renvoyé dans le statut (variable ?).
Les options (comparateurs) ci-dessous sont utilisés.

Opérateurs logiques

Tests sur les chaînes de caractères
[<chaîne de caractères 1> **=** <chaîne de caractères 2> **]**
teste si <chaîne de caractères 1> et <chaîne de caractères 2> sont égales.
Attention : les [et] doivent impérativement être entourés de blancs ainsi que les primitives de comparaison.
Autres tests sur les chaînes de caractères
!= (différentes),
-n (non vide),
-z (vide)

Tests sur les valeurs numériques :

- **-eq** (égales),
- **-ne** (différentes),
- **-gt** (strictement supérieure),
- **-ge** (supérieure ou égale),
- **-lt** (strictement inférieure),
- **-le** (inférieure ou égale) ;

Tests sur les fichiers :

- **-d** (catalogue),
- **-f** (fichier),
- **-r** (permissions de lire le fichier),
- **-w** (permissions d'écrire le fichier),
- **-x** (permissions d'exécuter le fichier).

* **exit** <valeur entière> termine le processus et renvoie la valeur passée en paramètre comme code de retour.

Exemple 1 : le fichier *monscript1* a le contenu suivant :

```
Set 'ls'
for i in $*
do
if [ -d $i ]
then echo "$i est un repertoire"
fi
if [ $i = "toto" ]
then echo "toto trouve. Voulez-vous voir son contenu ?"
read rep
case $rep in
o | O ) cat $i;;
n | N ) echo "pas de visualisation du contenu de toto";;
* ) echo "vous repondez vraiment n importe quoi"
esac
fi
done
```

4.6 Fonctions

2 syntaxes pour la définition d'une fonction :

function nom-fct {
bloc d'instructions
}

nom-fct() {
bloc d'instructions
}

syntaxe de l'appel d'une fonction :

valeur_retour=nom-fct <parametres>

4.7 Les caractères spéciaux et leur fonction

4.7.1 Composants de commande de base

Caractère	Signification	Contexte	Exemple
Espace	Délémiteur de composant de commande	commande	ls -l f.c
-	Flag désignant une option de commande	Commande	ls -al
/	Nom du répertoire root	Commande	cd /
/	Délémiteur de repertoires dans un chemin	Commande	cd /home/etudiant/user1
****	Inhibe le sens d'un métacaractère	Commande	echo *
&	Éxécution d'une commande en background	Commande	emacs &
;	Exécute séquentielle de commandes	Commande	date ; ls -l
()	Éxécute de commande(s) en sous shell	Commande	(date ; ls -l)

4.7.2 Redirection d'entrée sortie

Caractère	Signification	Contexte	Exemple
|	Tube	Commande	who | wc -l
|&	Erreur standard du tube	Commande	cc f.c -o f |& wc -l
>	Redirige la sortie standard	Commande	ls -lR >fic
>>	Redirige la sortie standard avec ajout ds fic	Commande	ls -lR >>fic
2>	Redirige la sortie standard erreur	Commande	cc f.c -o f 2>err
<	Redirige l'entrée standard depuis fichier	Commande	mail user@iup.univ-evry.fr <fic.c

4.7.3 Métacaratères sur nom de fichier

Caractère	Signification	Contexte	Exemple
~	Chemin du repertoire HOME de l'utilisateur	Commande	cp f.c ~/src
.	Lien par défaut avec le repertoire courant	Commande	cp ~/src/f2.c .
..	Lien par défaut avec le repertoire père	Commande	cd ..
*****	Représente une chaîne de caractères	Commande	cp ~/src/*.c .
?	Represente un caractère qcque	Commande	rm ~/bin/*.?
[ ]	Remplace un des caractères compris entre les bornes	Commande	rm ~/bin/f[0-9].o
[,]	Remplace un des caractères de la liste entre crochets	Commande	rm ~/bin/*.[o,b]
' '	Englobe une chaine de caractères sans substitution	Commande	echo '*'
" "	Englobe une chaine de caractères avec substitution	Commande	echo "*"

4.7.4 Composants généraux de scripts

Caractère	Signification	Contexte	Exemple
#	Délémiteur de commentaire	Script sh	# script.sh …
#!	Déclaration de l'exécution d'un shell	Script sh	#! /bin/sh
` `	Exécution de commande	Script sh	echo 'date'
$	Accès à une variable	Script sh	echo $USER
$$	N° du shell courant	Script sh	echo $$
$#	Nombre d'arguments du script	Script sh	if [$# -gt 0] then
$0 … $9	Les 10 premiers arguments du script	Script sh	echo $0
$*	Tous les arguments du script	Script sh	

4.7.5 Opérateurs mathématiques

Caractère	Signification	Contexte	Exemple
=	Affectation	Script sh	Var=2
* , / , %	Multiplication, division, modulo	Script bash	let a=$b*$c
+ , -	Addition , soustraction/operateur unaire	Script sh	
<< , >>	Décalage	Script bash	let d=$c<<2;echo d
&	ET bit à bit	Script bash	
|	OU bit à bit	Script bash	
^	OU EXCLUSIF bit à bit	Script bash	
~	NOT	Script bash	

4.7.6 Opérateurs d'affectation

Caractère	Signification	Contexte	Exemple
+= , -=	Add ou Soustraction + Assignation	Script bash	
*=, /=, %=	Mul/Div/Modulo + Assignation	Script bash	
&=,^=,%=	AND/OR/XOR + Assignation	Script bash	
>>=, <<=	Décalage droit/gauche +assignation	Script bash	
>=	Supérieur ou égal	Script bash	
<	Inférieur	Script bash	
<=	Inférieur ou égal		

4.7.7 Opérateurs de comparaison

Caractère	Signification	Contexte	Exemple
~	Complément	Script bash	If [$a=$b] then
==	Égalité	Script bash	
!=	Inégalité	Script bash	
>	Supérieur	Script bash	
>=	Supérieur ou égal	Script bash	
<	Inférieur	Script bash	
<=	Inférieur ou égal		

4.7.8 Opérateurs booléens

Caractère	Signification	Contexte	Exemple
!	Négation	Script bash	
||	Ou logique	Script bash	If [$a<9 || $a>0] then
&&	Et logique	Script bash	
!(||)	NOR logique	Script bash	
!(&&)	NAND logique	Script bash	
^	XOR logique	Script bash	
!(^)	NOR Exclusif logique	Script bash	

5. Quelques commandes externes usuelles par thèmes

Contenu : quelques commandes usuelles par catégorie, puis classées par ordre alphabétique avec exemples d'utilisation .

Aide en ligne

man, info

Informations sur :

l'utilisateur : who, id,logname
le système et l'environnement : uname, tty
la recherche de commandes : whereis, which
les fichiers : diff, cat, grep, head, tail, file, more
les répertoires : ls, pwd, find , df, du
la date : date

Manipulation de fichiers et répertoires

cd, rm, rmdir, mkdir
chmod, cp, mv, ln
sort, touch

Manipulation de chaînes de caractères

cut, tr, echo

Processus

kill, ps, sleep

Compression, décompression, archivage

Compress, gzip, gunzip, tar

Communication distante

telnet, ftp

Le courrier électronique

mail

Avertissement : Ci-dessous sont représentés quelques exemples de ces commandes.
La syntaxe de toute commande **com** est :

com [-**options**] **parametres** éventuels

Pour plus de détails sur les options d'une commande , faire appel aux commandes man (**man com**) ou (**com --help**)

cat f.c
Liste le contenu du fichier de nom "f.c"

cd src
Se déplace dans le répertoire de nom "src"

chmod go+r f.c
Donne le droit de lecture(**r**) de "f.c" au groupe(**g**) et aux autres utilisateurs(**o**)

compress -f f.c
Compresse le fichier de nom "f.c", en écrasant un fichier préexistant s'il existe

uncompress f.c
Décompresse le fichie "f.c"

cp f.c f2.c
Crée le fichier de nom "f2.c" identique au fichier "f.c".

cp -r rep ~ Recopie recurssive (-r) du repertoire "rep" dans le repertoire d'accueil(~)

ls -l f1 | cut -d ' ' -f4 Affiche le propriétaire du fichier "f1". La commande cut extrait du résultat de "ls" le 4eme champ (-f4) . Chaque champ est délimité (-d) par un espace

date
Affiche la date courante

diff -c fichier1 fichier2
Compare "fichier1" et "fichier2" en affichant 3 lignes avant et 3 lignes après chaque différence

echo $PATH
Affiche le contenu de la variable "PATH"

echo "Bonjour"
Affiche le message : Bonjour

mail user@iup.univ-evry.fr <lettre.txt
Envoie le fichier "lettre.txt" à user@iup.univ-evry.fr

file f
Retourne le type du fichier "f" (ascii, exec, ps, pdf, …)

find "." -name f.c -print
Recherche récursivement et affiche le chemin menant au fichier "f.c" à partir du répertoire courant(.)

ftp aramis@iup.univ-evry.fr se connecte au serveur aramis envue d'un téléchargement de fichiers

grep -n -i "printf" f.c
Affiche, avec leurs numéros, les lignes du fichier "f.c" qui contiennent la chaîne "printf"

gzip -c f.c > f.c.gz
Compresse le fichier "f.c" en conservant le fichier d'origine.
Le fichier compressé est "f.c.gz".

head f.c
Affiche les 10 premières lignes du fichier "f.c"

id
Retourne le login, l'UID, le groupe et le GID Courants

kill -9 1290
tue le processus de numéro 1290

ln -s f.c f2.c
crée un lien symbolique sur le fichier "f.c",
de nom "f2.c"

logname
Retourne le nom de login de l'utilisateur

ls -l src
Donne le contenu du répertoire "src" en
donnant des informations sur les fichiers (taille, dates,
...)

ls -alR ~ Affichage récursif (-R) de tous les fichiers de l'utilisateur y compris les fichiers cachés (-a)

man chmod
Affiche le manuel d'utilisation de la commande
"chmod"

mkdir sources
Crée le répertoire "sources"

more f.c
Affiche, page par page, le contenu du fichier
"f.c"

mv f.c f2.c
Renomme le fichier "f.c" sous le nom
"f2.c"

mv f.c rep
Déplace le fichier "f.c" dans le répertoire rep

ps -aef
Sous système V, donne la liste de tous les processus(e)
actifs en affichant toutes les info. les concernant(-af)

pwd
Retourne la référence absolue du répertoire courant

telnet aramis
Établit une connexion sur la machine aramis

rm f.c
Supprime le fichier "f.c"

rm -r rep
Supprime tous les fichiers du repertoire "rep" ainsi que ce dernier

rmdir sources
Supprime le répertoire "sources"

sleep 10
Attend 10 secondes

sort f.c
Affiche le fichier "f.c", les lignes étant triées par
ordre lexicologique

tar -cvf sources.tar sources
Crée une archive, de nom "sources.tar", du
contenu du répertoire "sources"

tar -xvf sources.tar sources
Extrait le contenu d'une archive, de nom
"sources.tar", dans le répertoire"sources".
L'écran affiche les noms des fichiers au fur et à
mesure de leur désarchivage.
L'archive est conservée.

touch f.c
Modifie la date de dernier accès au fichier
"f.c" à la date courante, si f.c n'existe pas il est crée

echo "Bonjour" | tr 'a-z' 'A-Z' La commande tr remplace les minuscules par les majuscules

ls -l f.txt | tr -s ' '
La commande tr substitue à chaque tabulation (-s) un espace

tail -5 f.c
: Affiche les 5 dernières lignes du fichier "f.c"

tty
: Donne le nom du terminal

uname -a
: Affiche, dans l'ordre, le nom du système d'exploitation, le nom et la release du système de la machine

whereis gcc
: Retourne le chemin complet d'accès à tous les répertoires contenant la commande gcc

which gcc
: Retourne le chemin complet d'accès à la commande gcc

who (am i)
: Retourne le nom des utilisateurs connectés sur cette machine

6. Liste des exercices des travaux dirigés

TD1 : Les commandes externes du SHELL

But : L'objet de ce TD est l'étude des commandes externes du SHELL et certains méta caractères

Prérequis :La connaissance de la syntaxe des principales commandes externes du SHELL

Exercice 1.1 :

Soit un répertoire contenant les fichiers suivants :
f1.c , f2.c , f3.f , f4.c , f5.f , f10.a , f11.a , a.out , e.c , t.f
Utiliser les méta caractères de substitution pour lister les fichiers suivants : (on recherchera l'ecriture minimale)

- les fichiers fortran (suffixe f),
- les fichiers C et fortran,
- les fichiers commençant par la lettre f,
- les fichiers C commençant par la lettre f,
- les fichiers dont le nom contient un chiffre avant '.',
- les fichiers dont le 2nd caractère est un chiffre,
- les fichiers dont le 2nd caractère est un '.' .

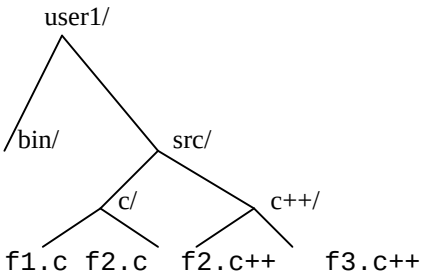
Exercice 1.2 :

Donner la commande qui liste l'ensemble des fichiers du répertoire "*/usr/sbin"* dont le nom commence par i suivi d'un caractère quelconque puis d'un point '.' puis de 2 lettres suivies de la lettre 'l', d'un 'm' ou d'un 't' et qui se terminent par un 'd'.

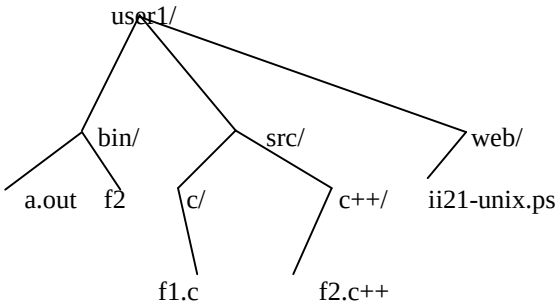
Exercice 1.3 :

Donner une suite de commandes SHELL qui permettent de faire passer l'arborescence de "user1" de l'état i à l'état i+1. On suppose que user1 se trouve dans son "Home Directory" :

Etat i :



Etat i+1 :



Exercice 1.4 :

Dessiner l'état i+2 du "Home Directory" de user1 à l'issue de l'exécution des commandes suivantes :
$ mv src/c/f1.c src
$ rm -r src/c
$ mv src/c++/f2.c++ src
$ cc src/f1.c -o bin/f1

TD2 et 3 : Les commandes externes du SHELL

Exercice 2.1 :

Donner la signification des commandes suivantes :
$ sort f.c | head
$ grep printf f.c | wc -l
$ grep printf f.c > sortie
$ cat /etc/passwd | grep user1

Exercice 2.2 :

Déterminer les commandes qui permettent de :
- savoir si l'utilisateur "user1" est connecté,
- afficher le nombre d'utilisateur du système ,
- afficher la liste des utilisateurs par ordre alphabétique,
- connaître le nombre de processus de "user1",
- connaître le nombre de processus de "root"

enregistrer dans le fichier "fuser1" la date et l'ensemble des fichiers de "user1".

Exercice 2.3 :

Ecrire le diagramme SHELL à base de fonctions systèmes (fork, exec, wait, exit) correspondant à chacune des commandes suivantes :
$ sort f.c | head
$ ps -aef | grep root | wc -l

Exercice 2.4 :

2.4.1
Ecrire une commande qui affiche l'ensemble des processus dont vous n'êtes pas propriétaire (votre nom d'utilisateur se trouvant dans la variable d'environnement $USER).
2.4.2
Donner la syntaxe qui lance la commande "sleep" en arrière plan("background") pendant une durée de 5 minutes.
2.4.3
Créer un sous-répertoire de "/tmp" ayant pour nom votre nom de login. Positionnez vous dans "/tmp". Créer dans le sous-répertoire précédent un fichier qui est la copie conforme de votre fichier ".profile" . Ce nouveau fichier doit avoir un nom ayant pour préfixe ".profile" et pour suffixe votre nom d'utilisateur. Protégez le contenu de ce sous-répertoire contre tout regard indiscret y compris le votre.
2.4.4
Ecrire une ligne de commande qui affiche uniquement le nom du port *TTY* courant (terminal) (sans utiliser la commande "tty")
2.4.5
Ecrire une commande "*find"* qui va rechercher à partir de votre répertoire HOME, les fichiers nommés "*core"* ou "*a.out"* et les supprimer.

TD 4 : Les scripts Shell - sh

But :

L'objet de ce TD est l'étude des fonctions des scripts sh.

Prérequis :

Une maîtrise des fonctions des scripts sh est requise.

Exercice 4.1 :

Ecrire un script sh qui réalise la somme de tous les arguments acquis a partir de la ligne de commande.

Exercice 4.2 :

Ecrire un script sh qui réalise l'affichage de tous les arguments de la ligne de commande.

Exercice 4.3 :

Ecrire un script sh qui réalise la copie d'un fichier sous plusieurs repertoires, utile pour l'administrateur. les repertoires file1 et file2 sont supposés exister, sinon fichier est duplique dans des fichiers ordinaires file1 et file2.

Exercice 4.4 :

Ecrire un script sh qui réalise l' affichage de la date en anglosaxon.

Exercice 4.5 :

Ecrire un script sh qui réalise l' affichage de la date en français

Exercice 4.6 :

Ecrire un script sh qui réalise l' Affichage d'un décompteur.

Exercice 4.7 :

Ecrire un script sh qui réalise la somme de tous les arguments acquis a partir de la ligne de commande

TD 5 : Les scripts Shell - sh

But :

L'objet de ce TD est l'étude des fonctions des scripts sh.

Prérequis :

Une maîtrise des fonctions des scripts sh est requise.

Exercice 5.1 :

Ecrire un script sh sous forme de fonction « recherche » qui recherche un nom passe en paramètre dans le fichier annuaire et qui affiche si le nom est trouvé ou pas.

Exercice 5.2 :

Ecrire un script sh sous forme de fonction « ajoute » qui ajoute un nom passe en paramètre dans le fichier annuaire et trie par ordre alphabétique » ce dernier.

Exercice 5.3 :

Ecrire un script sh sous forme de fonction « supprime » qui supprime un nom passe en paramètre dans le fichier annuaire.

Exercice 5.4 :

Ecrire un script sh sous forme de fonction « affiche » qui réalise Affichage de l'annuaire.

Exercice 5.5 :

Ecrire un script sh sous forme qui permet de créer un menu pour gerer un annuaire dans lequel se trouvent les fonctions recherche, ajoute, supprime et affiche déterminées précédemment.

TD 6 : Les scripts Shell - sh

But :

L'objet de ce TD est l'étude des fonctions des scripts sh.

Prérequis :

Une maîtrise des fonctions des scripts sh est requise.

Exercice 6.1 :

Ecrire un script sh "rep" qui affiche la liste des fichiers ordinaires lisibles(par tout utilisateur)se trouvant dans une liste de repertoires passés en paramètre. Si pas de paramètres, traiter le repertoire courant.

Exercice 6.2 :

Ecrire un script sh "rep2" qui à partir du repertoire courant va chercher tous les repertoires et sous repertoires. Pour chaque repertoire rencontré, afficher la liste des fichiers ordinaires qu'il contient(utiliser le script de l'exo. 1).

Exercice 6.3 :

Modifier le script "rep" de l'exo. 1 de façon à ce que celui-ci renvoie la liste des fichiers trouvés.

Exercice 6.4 :

Modifier le script "rep2" de l'exo. 2 de façon à ce que celui-ci affiche pour chaque fichier son nom et son nombre de lignes..

Exercice 6.5 :

Ecrire un script sh qui renomme plusieurs fichiers passes en parametre. Les nouveaux noms ont pour préfixe le 1er paramètre et pour suffixe le rang de renommage.

7. Liste des exercices des travaux dirigés

Université d'Evry-Val d'Essonne
IUP Sciences et Technologie
II21 - Unix
TP1

SUJET : Les commandes externes du SHELL

But :

L'objet de ce TP est l'étude des commandes externes du SHELL et certains méta caractères

Prérequis : La connaissance de la syntaxe des principales commandes externes du SHELL

PARTIE A : Gestion de Fichiers

Question A.1 :
Soit un répertoire contenant les fichiers suivants que vous aurez crées: f1.c , f2.c , f3.c, f4.c++ , f5.c , f10.c++ , f11.c++ , a.out , e.c , t.c++

Utiliser les méta caractères de substitution pour lister les fichiers suivants : (on recherchera l'ecriture minimale)
- les fichiers C++ (suffixe c++),
- les fichiers C et C++,
- les fichiers commençant par la lettre f,
- les fichiers C commençant par la lettre f,
- les fichiers dont le nom contient un chiffre avant '.',
- les fichiers dont le 2nd caractère est un chiffre,
- les fichiers dont le 2nd caractère est un '.' .

Question A.2 : (connaissance requise : redirection, pipe)
Donner la signification des commandes suivantes :

```
$ sort  |  head
$ grep printf  f.c | wc -l
$ grep printf f.c > sortie
$ cat /etc/passwd | grep user1
```

Question A.3 : (connaissance requise : commandes externes)
Déterminer une suite de commandes SHELL qui permettent de faire passer l'arborescence de votre repertoire d'accueil
de l'état i à l'état i+1 (cf. figure). On suppose que le reperoire courant est le même que le répertoire d'accueil.

Dans l'etat i , votre repertoire courant contient les fichiers de l'exo. 1 .
Dans l'etat i+1, 3 repertoires doivent être crées : src_c pour les fichiers C, src_c++ pour les fichiers C++, et bin pour
a.out et les executables correspondants aux sources précedents.

Question A.4 : (connaissance requise : commandes externes)
Dessiner l'arborescence de votre répertoire d'accueil suite à l'execution des commandes suivantes :

```
$ mv src_c/f2.c src_c++/f2/c++
$ rm -r bin
```

PARTIE B : Gestion de Processus

Question B.1 : (connaissance requise : commandes ps et who)
Déterminer les commandes qui permettent de :
- si l'utilisateur "user1" est connecté,
- le nombre d'utilisateur du système ,
- affiche la liste des utilisateurs par ordre alphabétique,
- le nombre de processus de "user1",
- le nombre de processus de "root"
- enregistre dans le fichier "fuser1" la date et l'ensemble des fichiers de "user1".

...

Université d'Evry-Val d'Essonne
IUP Sciences et Technologie
II21-Unix

TP 2 : Les scripts Shell - sh ou bash

But :

L'objet de ce TD est l'étude des fonctions des scripts sh ou bash.

Prérequis :
Une maîtrise des fonctions des scripts sh ou bash est requise.

Question 1 :
Ecrire un script sh sous forme de fonction qui recherche un nom passe en paramètre dans le fichier annuaire

Question 2 :
Ecrire un script sh sous forme de fonction qui ajoute un nom passe en paramètre dans le fichier annuaire et trie
par ordre alphabetique ce dernier.

Question 3 :
Ecrire un script sh sous forme de fonction qui supprime un nom passe en paramètre dans le fichier annuaire.

Question 4 :
Ecrire un script sh sous forme de fonction qui réalise Affichage de l'annuaire.

Question 5 :
Ecrire un script sh sous forme de fonction qui réalise creer un menu pour gerer un annuaire dans lequel se
trouvent les fonctions recherche, ajout, supprime et affiche déterminés précédemment.
.

Question 6 : modifier la fonction supprime de facon a ce qu'elle puisse supprimer un abonne parmi plusieurs
ayant le meme nom, en fournissant le prenom de celui-ci.

question 7 : modifier la fonction ajoute de maniere a ce que celle-ci puisse ajouter un abonne en precisant
son nom, prenom et numero de telephone..

Université d'Evry-Val d'Essonne
IUP Sciences et Technologie
II21-Unix

TP 3 : Les scripts Shell - sh ou bash

But :

L'objet de ce TD est l'étude des fonctions des scripts sh ou bash.

Prérequis :
Une maîtrise des fonctions des scripts sh ou bash est requise.

Question 1 :
Ecrire un script sh "rep1.sh" qui donne la liste de tous les fichiers ordinaires accessibles par tout utilisateur dans
une liste de repertoires passes en parametre. Si pas de parametre, traiter le repertoire courant.

Question 2 :
Ecrire un script sh "rep2" qui donne tous les sous repertoires (recurssivement) du repertoire passe en parametre.
"rep2" fera appel à "rep1"(qui devient une fonction) afin d'afficher la liste de tous les fichiers ordinaires
accessibles par tout utilisateur dans chaque sous repertoire.

Question 3 :
Modifier "rep2" en "rep3" de manière a ce que le nom et le nombre de lignes de chaque fichier ordinaire renvoyé
par "rep1" soient affichés. "rep1" nécessite également une modification qui consiste à renvoyer la liste des
fichiers ordinaires accessibles.

...