Département INFO

Rappels de cours et Travaux Pratiques Linux N°1/4
Introduction, système de fichiers

1 Un peu d'histoire

Le système **UNIX** est né dans les **Laboratoires Bell** en 1969. D'abord écrit par **Ken THOMPSON,** le développement du **langage C par Dennis RITCHIE** a permis une réécriture de ce système d'exploitation en une version plus portable.

Les universités se sont vite intéressées au produit et en ont acquis des licences. Une première version a été commercialisée sous le nom de **UNIX V7.** A partir de cette première version, **UNIX** s'est divisé en deux principales familles:

- ATT et les **laboratoires Bell** ont développé les versions **SYSTEM V;**
- **l'université de Berkeley** a développé les versions BSD.

D'autres développements ont donné naissance à plusieurs autres types de ce système plus ou moins inspirés des précédents : X*ENIX, VENIX, SCO UNIX, MINIX, XINU, HP- UX, Digital* Unix, ... Plus récemment, en 1991, un étudiant finlandais nommé **Linus B. Torsvald** développe son propre système d'exploitation et mets les source à disposition sur Internet. **Linux** (Linux Is Not UniX) est né. De nombreux contributeurs participent alors à corriger et enrichir le noyau. Complété par les applications GNU, le noyau permet de proposer un système d'exploitation complet et porté sur la plupart des plateformes matérielles actuelles.

2 Les principales caractéristiques de Linux

UNIX et Linux sont des systèmes d'exploitation **multi-utilisateurs, multi-tâches et multi-processeurs. Linux** est de plus **multi-plates-formes** et propose :

- un support des communications entre les processus (IPC, tubes, socket);
- une gestion des signaux processus;
- une gestion des terminaux suivant la norme **POSIX;**
- une gestion de la mémoire à la demande;
- un chargement dynamique des bibliothèques;
- un support de nombreux systèmes de fichiers, dont ses propres systèmes de fichiers **ext2 et ext3;**
- un support des protocoles réseaux courants dont **TCP/IP.**

2.1 Machine virtuelle

Le système d'exploitation et plus précisément le **noyau** propose une interface entre l'utilisateur et les ressources physiques de la machine. Seul le noyau est autorisé à accéder aux ressources matérielles. Le système d'exploitation est rendu ainsi plus stable car il évite des erreurs de l'utilisateur sur la gestion des ressources systèmes.

2.2 Le multi-tâches

Plusieurs utilisateurs peuvent travailler en même temps et chacun peut exécuter plusieurs processus de manière concurrente :

- chaque utilisateur et chaque processus du système est identifié; - le noyau fonctionne en temps partagé et gère des priorités.

Chaque processus est donc exécuté chacun son tour très rapidement (méthode du tourniquet ou "round robin").

2.3 Mode noyau et mode utilisateur

Un processus peut être exécuté dans deux différents modes. Dans le **mode noyau,** le processus a accès à l'ensemble des ressources de la machine sans aucune restriction. En **mode utilisateur,** le processus ne peut accéder qu'aux données qui lui ont été affectée.

Lorsqu'un utilisateur lance un programme, le processus créé s'exécute dans le mode utilisateur. Dès que le processus a besoin d'accéder aux ressources de la machine (allocation de mémoire, accès à un périphérique ...), un **appel système** est lancé vers le noyau. Ce dernier traite la requête et envoie le résultat au processus appelant.

2.4 Gestion des ressources

La gestion des ressources de la machine (disque, mémoire, réseau, ...) est faite sans blocage du système : si un programme exécuté par un utilisateur effectue des accès non autorisé à la mémoire (par exemple), seul ce programme sera interrompu et le système d'exploitation restera stable. Le système d'exploitation segmente et protège les données de chacun et empêche ainsi tout risque de conflit.

2.5 Le tout fichier

L'accès aux périphériques et aux ressources du système s'effectuent à travers des descripteurs de fichier. Sous **UNIX, "tout est fichier"** : un disque, la mémoire, la carte réseau (pas sous Linux), un lecteur de disquette, un lecteur de bande, un terminal, les entrée/sorties des processus....

3 Organisation du système de fichiers

Le système de fichiers de Linux est de type ext2. Il est organisé d'une manière arborescente comme le montre la figure 1

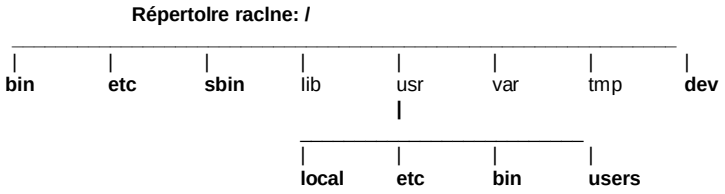

Fig. 1 - Système de fichiers

Parmi les nombreux répertoires, on trouve :

- **/bin** contient les commandes de base;
- **/usr/bin** contient la suite des commandes;
- **/tmp** contient les fichiers temporaires;
- **/etc** contient les fichiers de configuration du système;
- **/sbin et /usr/sbin** contiennent les commandes d'administration du système;
- **/dev** contient tous les fichiers spéciaux permettant d'accéder aux périphériques;
- **/home** contient les répertoires des utilisateurs.

4 Les fichiers

Les fichiers sur une partition ext2 ou ufs/ffs peuvent avoir un nom composé de 255 caractères. Le système étant multi-utilisateur, il doit identifier le propriétaire du fichier. De plus, les utilisateurs pouvant être divisés en plusieurs groupes, le fichier possédera une information concernant ce groupe d'appartenance. Afin qu'un utilisateur ne puisse pas supprimer ou accéder à un fichier ne lui appartenant pas, des droits sont appliqués aux fichiers. Le contenu du répertoire racine est le suivant :

```
drwxr-xr-x  2 root     root         4096     Oct 27 19:03         bin
drwxr-xr-x  3 root     root         4096     Oct 27 19:03         boot
drwxr-xr-x 17 root     root         77824    Oct 30 20:40         dev
drwxr-xr-x 37 root     root         4096     Oct 30 20:39         etc
drwxr-xr-x  3 root     root         4096     Oct 27 17:25         home
drwxr-xr-x  2 root     root         4096     Jun 21 20:32         initrd
drwxr-xr-x  7 root     root         4096     Oct 27 19:12         lib
drwxr-xr-x  2 root     root         16384    Oct 27 18:58         lost+found
drwxr-xr-x  4 root     root         4096     Oct 27 17:18         mnt
drwxr-xr-x  2 root     root         4096      Aug 23 1999         opt
dr-xr-xr-x 62 root     root         0         Oct 30 2001         proc
drwxr-x---  9 root     root         4096     Oct 30 21:36         root
drwxr-xr-x  2 root     root         4096     Oct 27 19:08         sbin
drwxrwxrwt Il root     root         4096     Oct 30 21:35         tmp
drwxr-xr-x 15 root     root         4096     Oct 27 19:03         usr
drwxr-xr-x 15 root     root         4096     Oct 27 19:00         var
```

La dernière colonne donne le nom du fichier. La première donne le type du fichier ainsi que ses attributs (lecture, écriture, exécution) pour le propriétaire, le groupe et les autres. La troisième donne le propriétaire du fichier. La quatrième colonne donne le groupe d'appartenance. Les suivantes donnent la taille du fichier et sa date.

4.1 Les fichiers ordinaires

Les fichiers ordinaires ont le premier caractère de leurs attributs égal à un tiret. Ces fichiers peuvent être de type texte ou binaire. Les attributs donnent ou non un accès en lecture (**r**), en écriture (**w**) ou en exécution (**x**).

4.2 Les répertoires

Les répertoires sont identifiés dans l'attribut parun "d". Ces fichiers possèdent aussi des droits en lecture et écriture. Par contre, le caractère "x" donne un droit de passage. Une commande cd pourra alors être rendu impossible sur un répertoire.

4.3 Les fichiers lien

Les fichiers lien permettent de référencer un fichier ordinaire ou un répertoire en utilisant un nom différent. Si par exemple, vous avez le répertoire **/usr/local/bin** et que vous ne voulez pas avoir à retaper à chaque fois la totalité pour vous placer dans celui-ci, il est possible de créer un fichier "raccourci".

Les fichiers lien sont de deux types. Les **liens simples** ont le premier caractère de l'attribut égal à un **tiret.** Les **liens symboliques** ont quand à eux un "l" comme premier caractère de leur attribut. Les droits du fichier référencé deviennent les droits du fichier lien.

4.4 Les fichiers spéciaux

Les fichiers spéciaux permettent d'accéder aux périphériques du système : disques, disquettes, lecteur de bandes, carte réseaux...

Toutes les opérations de création, d'effacement et de gestion des fichiers de périphérique (fichiers spéciaux) ne peuvent être effectuées que par l'administrateur. Un simple utilisateur ne pourra accéder aux périphériques qu'en respectant les droits que possèdent les fichiers leur correspondant.

Un fichier spécial a le premier caractère de son attribut qui correspond à la lettre **c** ou **b** suivant qu'il soit de type caractère ou bloc.

5 Connexion sur le système

UNIX étant **multi-utilisateurs,** vous devez vous connecter sur le système en vous identifiant à travers un nom et un mot de passe. Une fois connecté, vous vous trouvez dans le répertoire vous appartenant. On nomme celui-ci **Home Directory.**

Un programme nommé **shell** a été lancé lors de votre connexion. Le shell est l'interface entre le système et vous. Il interprète les commandes tapées par l'utilisateur et envoie au noyau les requêtes nécessaires à leur action. Plusieurs types de shell existent : le **Bourne** shell, le C shell, le **Korn** shell ou encore le **Bash** shell. C'est ce dernier qui est exécuté pour votre connexion. Vous pouvez obtenir ces caractéristiques avec la commande **man bash**

Références

[1] Programmation Linux 2.0 "API Système et fonctionnement du noyau"
Par Rémy Card, Erie Dumas et Franck Mével, Eyrolles

[2] La programmation sous UNIX - 3éme édition
Par Jean-Marie Rifflet, Ediscience International

Exercices

Si vous souhaitez de plus amples informations sur les commandes UNIX, vous pouvez utiliser le manuel en ligne ***man ou info*** *suivi du nom de la commande ou du nom de votre shell.*

Premier contact

Connectez-vous sur le système. Vous pouvez afficher l'endroit où vous vous trouvez avec la commande pwd. Le déplacement dans le système de fichiers s'effectue avec la commande cd (change directory). *Placez-vous au sommet de l'arborescence (/).*

Trouvez des options à la commande Is afin de visualiser les fichiers commençant par un "." ainsi que les informations concernant les droits, propriétaire et type de fichiers. ***Commentez ce que vous visualisez.***

Les fichiers ordinaires

La création et la visualisation

Une des manières simple de créer un fichier ordinaire est d'utiliser la commande cat. *Tapez la commande suivante*

cat > ou

Le système effectue un retour à la ligne. *Tapez alors le texte* ***clear;pwd*** *et terminez la saisie par* ***CTRL-D.***

Vous venez de créez un fichier du nom de ou. *Vérifiez qu'il a bien été créé avec la commande Is. Remarquez que le premier caractère de l'attribut de ce fichier est un tiret ce qui signifie que le fichier est bien ordinaire.*

Vous pouvez réutiliser la commande cat pour connaître le contenu du fichier. *Tapez cat ou*

D'autres commandes comme **more ou less** sont possibles. Leur avantage par rapport à cat est que si votre fichier est très long, l'affichage du contenu se fera page par page. *Vérifiez ce fait en visualisant le contenu du fichier* ***/etc/passwd.***

La copie

La copie de fichier se fait avec la commande cp. *Copiez le fichier ou en un deuxième fichier nommé **where.** Recommencez l'opération une seconde fois. Remarquez que le système ne vous signale pas que vous risquez d'écraser le fichier **where.** Trouvez une option qui vous demande une confirmation.*

La comparaison

La commande permettant de comparer 2 fichiers est diff. *Créez un second fichier nommé ou2 qui contient le texte suivant : pwd;clear et comparez celui-ci avec le fichier ou.* Le résultat vous donnera les lignes qui diffèrent entre les fichiers.

L'affichage des fichiers

Vous avez déjà utilisé plusieurs fois la commande Is avec l'option 1 qui permet d'afficher les caractéristiques des fichiers. Cette commande possède de nombreuses options.

Votre **shell** permet l'emploi de **métacaractères.** Ceux-ci permettent de remplacer une partie du nom des fichiers : *, ?, {}**,** [ ]**.** Nous les verrons plus précisément lors du prochain TP. Retenez juste pour l'instant que si vous voulez lister tous les fichiers commençant par ou, vous taperez
Is ou*.

La suppression

Pour supprimer des fichiers, la commande rm est employée. La commande rm ne demande pas une confirmation avant de supprimer le fichiers. Cependant, certains shells permettent d'obliger une confirmation. Si *ce n'est pas votre cas, cherchez une option pour la commande rm afin qu'elle demande confirmation et supprimez le fichier ou2.*

Le déplacement et le changement de nom

Pour déplacer un fichier ou pour le renommer (renommer est en fait déplacer le fichier au même endroit en changeant son nom ...), vous utiliserez la commande **mv**.

Renommez le fichier ou en un fichier nommé OU. Vous noterez au passage que le système UNIX différencie les majuscules des minuscules.

La mise à jour de la date et de l'heure

La commande **touch** permet de mettre à jour la date d'un fichier. Si ce fichier n'existe pas, la commande crée un fichier vide (cela permet ainsi de créer rapidement des fichiers). *Regardez la date et l'heure du fichier where et mettez la à jour.*

Le changement des caractéristiques

La modification du propriétaire et du groupe est faite respectivement par les commandes **chown et chgrp.** Concernant les droits, ceux-ci se divisent en 3 parties (voir figure 2)

Fig. 2 - Droits des fichiers

Si la lettre apparaît, le droit correspondant est donné. Si la lettre est remplacée par un tiret, le droit correspondant est refusé. La mise à jour des droits se fait avec la commande . Deux manières de l'utiliser existent. La manière "standard" (que vous retrouverez dans le manuel en ligne) pour modifier les droits d'un fichier est :

- **chmod u+r OU** donne le droit en lecture (r) au propriétaire (user) pour le fichier **OU**
- **chmod u-r OU** interdit le droit en lecture (r) au propriétaire (user) pour le fichier **OU**
- **chmod uo+x OU** donne le droit en exécution (x) au groupe (u) et au autres (o) pour OU

Vous rencontrerez des fichiers dont les droits contiennent d'autres lettres que r, w et x. Vous trouverez les lettres s, S, t. Nous en reparlerons plus tard. Si vous êtes impatients, consultez le manuel en ligne de la commande **chmod.**

Essayez d'exécuter le fichier OU. Que se passe-t'il ? Modifiez les droits du fichier OU afin que tout le monde puisse l'exécuter et le lire et que seul le propriétaire puisse le modifier (écrire). Vérifiez qu'il est maintenant possible d'exécuter le fichier OU.

La recherche d'un fichier

La commande find est utilisé pour rechercher des fichiers parleur nom, leur taille, leur date, leur type etc... Cette commande est très puissante. Elle permet entre autre d'associer une action aux fichiers qui seront trouvés. La syntaxe de base de la commande est la suivante

find ***répertoire-de départ caractéristiques- recherchés***

On peut donner comme exemple :

- **find -name OU** recherche à partir du répertoire / tous les fichiers dont le nom est **OU**
- **find -newer OU** recherche à partir du répertoire courant (.) les fichiers dont la date est plus récente que celle du fichier OU

Il est possible d'utiliser les caractéres ***, ?,[]** pour donner des caractéristiques communes aux fichiers recherchés :

- **find / -name wh?r[a-i]*** recherche à partir de / les fichiers qui commencent par wh dont la troisième lettre est quelconque et la quatrième dans l'intervalle a à i avec toutes les lettres suivantes quelconques.

Les répertoires

Vous utiliserez la commande **mkdir** pour créer un répertoire et la commande **rmdir** pour le supprimer. Il faut noter que **rmdir** ne supprime le répertoire que si celui-ci est vide.

Toutes les autres opérations déplacement, mise à jour de la date et des caractéristiques, recherche, changement du nom sur les répertoire sont effectués avec les commandes utilisées pour les fichiers simples.

Concernant les droits, le "x" indiquant un droit en exécution pour un fichier ordinaire devient un droit de passage.

Création et modification

Sous votre Home directory, créez un répertoire "rep" en utilisant la commande ***mkdir.*** *Constatez que le premier caractère des attributs est "d".*

Déplacez le fichier OU dans ce répertoire en utilisant mv. Vérifiez avec la commande `ls` *le bon déroulement de ce déplacement. Modifiez les droits du répertoire afin d'éviter que tout autre utilisateur que vous-même puisse y passer.*

Les fichiers lien simple et lien symbolique

La création

La création d'un lien simple se fait avec la commande ln de la manière suivante

```
ln fichier-référencé nouveau-fichier
```

La création d'un lien symbolique se fait avec la même commande en ajoutant l'option s. La syntaxe est la suivante :

```
ln -s fichier-référencé nouveau-fichier
```

La création et la suppression

Recopiez le fichier OU dans votre répertoire d'origine. Tapez la commande "In OU lien". Le système a créé un fichier de type lien nommé lien. Visualisez ce fichier. Que constatez-vous ? Quel est le premier caractère de l'attribut de ce fichier ?

Rajoutez du texte au fichier lien en tapant la commande ***cat >>lien*** *et terminez la saisie par* ***CTRL-D.*** *Que constatez vous au niveau du fichier OU? Et si vous ajoutez du texte à OU, que se passe-t'il pour le fichier lien ? Si vous effacez le fichier OU, que devient le fichier* ***lien*** *? Expliquez ce qui se passe, à votre avis, lors d'une commande In.*

Revenez à la configuration de départ (cad le fichier OU dans votre répertoire d'origine où vous vous placez). Tapez la commande "In -s OU lien-s " et suivez le même raisonnement que précédemment sachant que vous venez de créer un fichier de type lien symbolique.

Pour les plus rapides...

Utilisez la commande **mount** sans aucun paramètre pour connaître l'organisation de votre système de fichiers. Utilisez la commande df pour :

1. connaître l'occupation en blocs de votre système de fichier;

2. connaître le nombre total, actuel et libre de fichiers.

Faites le lien entre ces résultats et votre cours sur les systèmes de fichiers.

Rappels de cours et Travaux Pratiques Linux N°2/4

Première partie : Le Shell: caractéristiques et paramétrage

1 Notion de shell

Le shell est l'interface entre le système et vous. Il est lancée lors de votre connexion et pour chaque terminal graphique que vous ouvrez. Il interprète les commandes que vous tapez en respectant une grammaire bien précise. Comme nous l'avons dit dans le précédent TP, l'utilisateur possède un shell par défaut. Il lui est possible de choisir un shell différent avec la commande **chsh.**

Les shells acceptés par le système sont listés dans le fichiers **/etc/shells.** Parmi les shells possibles, vous aurez le choix entre le **Bourne Shell (sh), le C Shell (esh), le Korn Shell (ksh)** ou encore le **Bash Shell (bash).** Chacun possède des caractéristiques propres et interprète les commandes différemment. Dans la suite de ce document nous étudierons le **Bash.**

2 L'éditeur de fichiers emacs

2.1 Quel éditeur utiliser?

Afin de pouvoir paramétrer votre Shell, vous allez avoir besoin d'un éditeur de fichier. Sous tous les systèmes **UNIX,** vous trouverez l'éditeur nommé vi. Ce dernier, outre le fait qu'il soit universel, est très complet et puissant. Il est toutefois assez difficile de l'utiliser lorsqu'on débute. Toutefois, nous conseillons de revenir plus tard sur cet éditeur et de connaître son utilisation de base. Dans certains cas, vous pouvez ne pas avoir le choix et être ainsi obligé de l'utiliser. De plus, les commandes utilisées dans l'éditeur vi sont reprises par d'autres commandes systèmes (ed, sed, par exemple).

Dans un premier temps et afin d'éviter de consacrer une séance à l'apprentissage de vi, nous préférerons parler de **emacs.** Ce dernier est un éditeur qui n'est pas par défaut dans un système **UNIX.** Il est souvent ajouté dans le système par l'administrateur car bon nombre de gens savent l'utiliser. Il est complet, puissant et simple d'emploi.

D'autres éditeurs existent et s'inspirent plus ou moins des 2 précédents. Lorsque vous travaillez sur des stations graphiques, vous pourrez aussi utiliser des éditeurs graphiques (xemacs, nedit, kedit ...) mais il reste bon de connaître un éditeur non-graphique pour ne pas être bloqué lorsqu'on se retrouve devant un terminal texte.

2.2 Utilisation de emacs

Quelques commandes suffisent pour travailler sous **emacs.** Le lancement du programme se fait en tapant **emacs** suivi ou non d'un nom de fichier. Ensuite, suivant l'action souhaitée, vous utilisez des combinaisons de touches. Quelques unes de ces combinaisons vous sont présentées dans le tableau 1 (CTRL représente la touche Control).

Combinaison	Action
CTRL-x,CTRL-f	Lecture d'un fichier
CTRL-x,CTRL-i	Insertion d'un fichier
CTRL-x,CTRL-s	**Sauvegaxde** d'un fichier
CTRL-S	Recherche d'une expression
CTRL-x,CTRL-c	Quitter Emacs

TAB. 1 - Quelques combinaisons de touches pour Emacs

Emacs permet le copier-coller, le multi-fenêtrage, l'aide en ligne, etc... Consultez le manuel en ligne et faites quelques essais afin de vous habituer à l'utilisation de l'éditeur.

3 Caractéristiques générales du Bash

3.1 Les commandes internes et externes

Les commandes que nous verrons tout au long de ces travaux pratiques se divisent en deux catégories :

1. les commandes internes;

2. les commandes externes.

Les commandes internes sont contenues dans le binaire du programme **bash.** La commande cd en est un exemple. Les commandes externes représentent des binaires indépendants du shell comme, par exemple, les commandes ls ou df

3.2 Les variables locales et d'environnement

Chaque shell possède deux zones de données : la zone de variable locale et la zone d'environnement. La définition d'une variable locale se fait tout simplement de la manière suivante

```
NOM-DE-VARIABLE=Valeur-de-la-variable
```

Créez les variables locales *NBRE* et *PHRASE* respectivement égales à 12 et *bonjour.*

Vous pouvez visualiser la valeur des variables en utilisant la commande **echo.** Pour voir, par exemple, la valeur de **NBRE,** vous taperez

```
echo $NBRE
```

La commande set permet de visualiser la totalité des variables locales. *Utilisez là. Vous* devez constater que le système a déjà initialisé quelques variables pour vous.

Les variables locales ne sont exploitables que par le shell courant. Si vous lancez un programme à partir de votre shell, le processus ne connaîtra pas les variables locales. Si vous souhaitez qu'une variable soit exploitée par d'autres programmes, vous devrez la placer dans la zone d'environnement grâce à la commande **export.** Cette commande s'emploie de la manière suivante :

```
export NOM-DE-VARIABLE
```

Vous trouverez quelques variables du shell dans le tableau 2.

EDITOR	**Fixe** l'éditeur par défaut
TERM	Indique le type de terminal utilisé
USER	Contient le nom de connexion que vous utilisez
PS1	Variable contenant le prompt affiché à l'écran
HOME	Indique votre Home Directory
PATH	Contient tous les chemins de recherche

TAB. 2 - Quelques variables d'environnement du shell

Visualisez les variables d'environnement avec la commande env et constatez que vos variables NBRE et PHRASE ne s'y trouvent pas. Exportez les deux variables et vérifiez qu'elles apparaissent comme variables d'environnement.

Notez que si vous vous déconnectez, ces variables seront perdues et devront être redéfinies à chaque connexion. Il est possible de supprimer une variable grâce à la commande **unset** suivie du nom de la variable. *Faites-le avec NBRE et vérifiez que cette variable n'apparaît plus.*

3.3 Les alias

Une des fonctionnalités apportées par le `bash` est la possibilité de définir des **alias.** Un **alias** permet de définir une commande souvent utilisée avec de multiples options à travers un nom court. La commande **alias** utilisée seule liste les "raccourcis" déjà définis. Un alias peut être supprimé avec la commande **unalias.** Pour définir un alias, vous taperez

```
alias Nom-Alias='Commande et options'
```

Depuis le début des TP, nous avons exécuté à de multiples reprises la commande Is -1. Afin de simplifier la frappe de cette commande, le système a défini pour vous l'alias ll. Mais celui-ci ne vous affiche pas les fichiers commençant par un ".". *Définissez un alias nommé la qui affiche la totalité des fichiers.*

Sachant que le bash définit une variable nommée OLDPWD qui contient, après une commande cd l'ancien répertoire où vous vous trouviez, créez un second ***alias*** *nommé back qui permet de revenir dans l'ancien répertoire.*

3.4 L'historique et le complément de commande

Le bash vous permet de rappeler les commandes que vous avez déjà tapées. Les touches **flèche-haut et flèche-bas** parcourent cet historique. Outre le fait de les rappeler, il est possible de modifier ces anciennes lignes de commande grâce aux deux autres flèches (droite et gauche).

Toutes les commandes que vous avez utilisées auparavant sont stockées dans **bash-history.** Lorsque vous vous déconnectez, le shell sauvegarde les commandes dans ce fichier.

Les variables d'environnement **HISTSIZE et HISTFILESIZE** fixent respectivement le nombre de commandes conservées en mémoire et le nombre de commandes conservées dans le fichier **bash-history.**

Une autre fonctionnalités du bash est de pouvoir compléter une commande ou un fichier grâce à la touche de **tabulation.** Si, par exemple, vous souhaitez taper une commande **alias,** vous commencerez par taper ali suivi d'une tabulation. Le shell, après une recherche des commandes commençant par ali complétera la commande. Dans le cas où plusieurs commandes commencent par ali, le terminal émettra un bip et vous pourrez obtenir la liste des commandes en réappuyant une **seconde fois** sur la **tabulation.**

De même, si vous souhaitez vous déplacer dans un répertoire, afficher un fichier ou une variable, le shell effectue le complément de nom.

3.5 Les métacaractères

Les métacaractères sont les caractères qui possèdent une signification particulière pour le shell. Nous en avons déjà rencontré avec l'emploi de la commande Is. Nous avions vu les caractères *, ?, [et]. En rappelant la fonction de ces derniers, nous allons en introduire de nouveaux dans le tableau 3.

Caractère	Fonction	Caractère	Fonction
*	Remplace un ensemble de ca-Ractères	#	caractère de commentaire
?	Remplace un caractère dans Une chaîne	‘	Délimitent une chaîne de ca-ractères et n'interprète pas les autres métacaractères
[et]	Permettent d'indiquer un en-Semble	"	Délimitent une chaîne de ca-ractères et interprètent les métacaractères $, \ , ' ,…
-	Séparateur d'un ensemble	`	Ce qui est encadré par les an-tiquotes est exécuté
~	Permet de référencer votre Home directory	\	Permet de ne pas interpréter le métacaractère qui suit
$	Permet de référencer une va-Riable		

TAB. 3 - Quelques métacaractères

Ces quelques exemples du tableau 4 vous aideront sûrement à mieux comprendre les règles de priorité :

Exemple	Résultat
Echo 'Je suis dans $PWD'	Je suis dans $PWD
Echo "Je suis dans $PWD"	Je suis dans /tmp

Echo "Je suis dans \$PWD"	Je suis dans $PWD
Echo "Nous sommes le `date`"	Nous sommes le Wed Jun 1996
Echo "Nous sommes le 'date' "	Nous sommes le 'date'

TAB. 4 - Règles de priorité des métacaractères

4 Paramétrage du shell

Lorsque vous vous connectez ou que vous lancez le shell de manière interactive (à la différence d'un script shell ou d'un terminal graphique), un programme **Bash** est lancé par le système. Celui-ci interprète plusieurs fichiers :

1. des fichiers généraux : **/etc/profile puis /etc/bashrc**
2. des fichiers personnels hébergés sous votre compte : **bash-profile et bashrc.**

Ces fichiers fixent des variables et exécutent des commandes visant, en général, à paramétrer votre shell. Quand vous quittez un shell interactif ou que vous vous déconnectez, le fichier **.bash-logout** est exécuté.

Le fichier **bash-profile** contient les variables d'environnement. Les alias et les commandes sont définis dans le fichier **bashrc.** Vous trouverez ci-dessous un exemple de ces deux fichiers

```
#############################################################################
# bash-profile
# Get the aliases and functions
if [ -f -/.bashrc ]; then
. ~/.bashrc              #Appel au fichier bashrc si il existe
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin     #surdéfinition de la variable contenant les chemins
                        #vers les commandes
BASH-ENV=$HOME/.bashrc
USERNAME="boebionil"

export USERNAME BASH-ENV PATH #les variables passent dans l'environnement
#############################################################################


############################################################################## # bashrc #
System wide functions and aliases # Environment stuff goes in bash-profile umask 077
#protection maxi pour la création de fichier alias back="cd $OLDPWD"
#############################################################################
```

Notez que pour que les changements dans ces fichiers soient pris en compte dans votre environnement, vous devez vous déconnecter et vous connecter à nouveau ou taper

```
bash_profile
```

Exercices

Connexion en mode non graphique

Connectez-vous en mode non graphique. Pour cela, appuyez sur la combinaison de touches **CTRL-ALT-Fl.** Vous devez obtenir une bannière de login.

Modification de votre environnement

A l'aide d'un éditeur de fichiers, modifiez vos fichiers d'environnement afin

- d'afficher à la connexion un message de bienvenue contenant votre nom de login et le type de machine (voir commande ***uname);***
- définir un alias vous permettant de revenir dans le répertoire précédent;

- définir un alias vous permettant de connaître la place qu'occupe vos fichiers dans votre compte (voir commande du);
- définir un alias vous permettant de trouver tous les fichiers de votre compte commençant par ". b".
- d'afficher le nombre de secondes pendant lesquelles vous avez été connecté au moment de la déconnexion.

Deuxième partie : Le Shell: gestion des processus, redirection d 'entrée/sortie

1 Définition

Un programme se compose d'un ou plusieurs fichiers composés d'instructions. Ces fichiers sont stockés sur un support comme un disque dur ou un CDRom. Un programme est une entité passive. Lorsque le programme est lancé, un processus est créé par le système d'exploitation. Un processus est une entité active et possède des caractéristiques (PID, PPID, UID ...) qui peuvent varier dans le temps (priorité, adresse de l'instruction suivante, par exemples).

2 L'exécution d'une commande UNIX dans un shell

La gestion des processus par le système d'exploitation **UNIX** est dite hiérarchisée. Lorsque vous tapez une commande externe au shell, (cad qu'un fichier binaire correspondant à la commande est chargé; par exemple : ls) un processus fils du shell est créé et la commande s'exécute dans celui-ci. Le processus père (le shell) attend que la commande soit exécutée pour redevenir actif Deux appels systèmes sont en fait utilisés : **fork et exec** et on peut modéliser la procédure avec la figure 1.

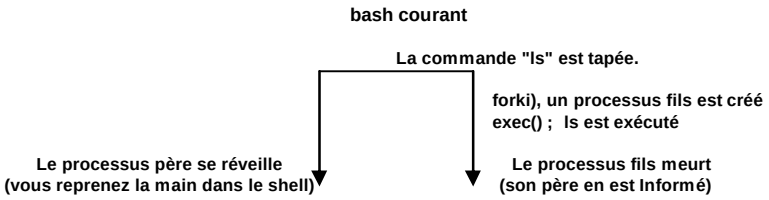

FIG. 1 - Lancement d'une commande

Il est possible d'obliger l'exécution d'un processus dans le shell courant. Pour cela, le lancement de la commande se fera à travers la commande interne **exec** : **exec sleep 10,** par exemple. De même, dans le cas d'un script, on peut utiliser le point ou la commande interne **source.** Par exemple, l'exécution du fichier **bash-profile** dans le shell courant se fera de la manière suivante **bash-profile**

3 Caractéristiques des processus

3.1 Caractéristiques générales

Tout comme les fichiers, un système **UNIX** identifie les processus grâce à un nombre. Cette information s'appelle le PID (Process IDentificator). Comme un fichier, un processus a un propriétaire. L'analogie s'arrête ici. Toutes les informations concernant les processus s'obtiennent avec la commande ps. Il existe une version System V et une version *BSD* de cette commande. Nous ne verrons que quelques options BSD (vous pouvez retrouver l'ensemble des options en consultant le manuel en ligne).

En tapant la commande ps axl, vous allez obtenir des informations concernant tous les processus :

- le propriétaire (UID); - le numéro du processus (PID); - le numéro du processus père **(PPID);** - la priorité (PRI); - la valeur de NICE (NI); - l'état **(STAT)** - le nom du device vers lequel il est dirigé (TTY); - le temps passé dans la cpu **(TIME);** - le nom de la commande exécutée **(COMMAND).**

Si vous ajoutez l'option "f, les liens de parenté entre les différents processus apparaîtront plus facilement. Notez que toutes les processus sont lancées par le processus **init** dont le **PID est 1.** Le processus père du processus **init** n'est autre que le noyau (PID égal à 0). Vous pouvez utiliser la commande **pstree** pour visualiser d'une seconde manière l'arborescence des processus.

Si vous n'êtes intéressé que par vos processus, la commande ps -fux vous affichera ceux-ci.

3.2 Etat d'un processus

L'état d'un processus peut être les suivants

- D le processus est ininterruptible; - R le processus est en train de s'exécuter; - S le processus est endormi; - T le processus est stoppé; - Z le processus est zombi : il est mort mais son père ne le sait pas.

D'autres informations peuvent aussi apparaître

- W le processus a été swappé; - < le processus possède une priorité haute; - N le processus possède une priorité basse; - L le processus possède des pages mémoires verrouillées.

3.3 Les entrée/sorties d'un processus

Chaque création de nouveau processus s'accompagne de la création d'une table de descripteurs de fichier. T~ois entrées sont créé par défaut - **STDIN, STDOUT et STDERR.** Ces descripteurs correspondent respectivement à l'entrée standard (numéro logique 0), à la sortie standard (numéro logique 1) et à la sortie d'erreur standard (numéro logique 2). Par défaut, ces fichiers sont affectés au clavier pour les entrées et à un TTY pour les sorties mais il est possible de les rediriger.

La redirection d'entrée se fait avec le symbole <, celle de sortie avec les symboles > et » et celle de sortie d'erreur avec les symboles 2 > et 2 ». Le tableau 1 vous donne quelques exemples possibles.

Commande	**Action**
mail user3 < fie	envoi du fichier fie à l'utilisateur user3 par un mail
ls >fie	envoi du résultat de la commande ls dans le fichier fic
date >>fie	ajout du résultat de la commande date dans le fichier fie
find / -name toto 2>/dev/null	recherche d'un fichier et envoi des erreurs retournées dans le fichier null (poubelle)
$ **cp *.* ~/tmp 2>>erreur.log**	ajout des erreurs retournées par cp dans le fichier erreur.log (archivage d'erreur)

TAB. 1 - Redirection d'entrée/sortie

Bien entendu, il est possible de cumuler plusieurs redirections

$ **find /home -name bash-profile >resultat 2>erreur**

3.4 Le code retour d'un processus

L'exécution de toute commande **UNIX** se termine par l'envoi d'un code retour au processus père. Celui-ci indique le bon déroulement ou non de la commande. En général, un code retour égal à 0 indique un bon déroulement alors quel indique une erreur de syntaxe et 2 une erreur d'emploi de la commande.

La variable $? du shell permet d'afficher le code retour de la dernière commande exécuté.

4 Gestion des processus

4.1 Background et Foreground

Vous avez remarqué que lorsque vous lancez une commande UNIX, vous ne récupérez la main que lorsque l'exécution de cette commande est terminée. Grâce au multi tâche, il est possible de faire exécuter celle-ci en tâche de fond et ainsi de récupérer le prompt immédiatement. Pour cela, vous utiliserez le caractère & à la fin de la commande. Toutefois, il faudra dans ce cas rediriger la sortie standard dans un fichier :

$ **find /home -name bash-profile > resultat.txt** &

La commande **jobs** vous permet d'obtenir la liste des processus en tâche de fond. Il est possible de passer un processus de l'arrière plan au premier plan avec la commande fg, de la stopper en tapant **CTRL-Z** ou encore d'utiliser la commande bg pour passer un job en background.

Le paramètre des commandes bg et fg est soit un PID soit un numéro retourné par la commande **jobs.** Dans ce dernier cas, le numéro sera précédé du symbole %.

4.2 Les signaux

Le système communique avec les processus à l'aide de signaux. Par exemple, si vous lancez une commande et que vous tapez les touches **CTRL-Z,** le processus en cours recevra le signal numéro 24 (SIGSTOP) et stoppera son traitement. Une déconnexion provoquera l'envoi du signal 1 (SIGHUP) à tous les processus. Lorsque vous tapez **CTRL-C,** vous envoyez un signal 2 (SIGINT) au processus courant.

Si un processus reçoit un signal auquel aucun traitement n'est associé, il meurt. Il est possible de définir ce traitement associé ou d'ignorer un signal. Ceci reste cependant impossible pour certains signaux.

Un utilisateur peut envoyer un signal à un processus avec la commande **kill.** Celle-ci envoie par défaut le signal numéro 15. La syntaxe est

$ **kill** -*numéro* PID
ou
$ **kill** -*nom-du-signal* PID

Vous pouvez aussi utiliser une syntaxe utilisant le symbole % suivi d'un numéro de processus issu de la commande **jobs.** Pour tuer un processus, la commande **kill** avec le **PID** comme seul argument suffit normalement. Afin d'être sûr d'éliminer un processus, l'envoi du signal 9 est requis.

4.3 Détachement d'un processus

Si vous lancez un long traitement sur une machine UNIX, le processus sera tué si vous vous déconnectez. Il existe un moyen de laisser s'exécuter un programme après une déconnexion et de visualiser les résultats stockés dans un fichier. Pour cela, vous utiliserez la commande **nohup** avant votre commande et vous redirigerez les résultats de votre processus dans une fichier.

4.4 La commande trap

Elle permet d'ignorer des signaux ou de leur associer un traitement particulier. Le tableau 2 donne quelques exemples. La syntaxe de cette commande est :

$ **trap** ***'commande' numéro-signal***

Commande	**Action**
$ trap ‘’ 2	ignore le signal 2
$ trap 2	restaure le traitement par défaut
$ **trap**	liste les signaux piégés
$ **trap 'echo bonjour' 2**	exécute echo bonjour dès réception du signal 2J

TAB. 2 - Exemples de la commande trap

4.5 Priorité des processus

Il est possible de jouer sur la priorité d'exécution des processus avec la commande **nice.** Celle-ci se place avant le programme à exécuter en précisant une valeur. Ces valeurs sont comprises entre 0 et 19 pour un simple utilisateur du système et entre -20 et 19 pour le super utilisateur. La valeur 0 correspond à la valeur par défaut, 19 à la priorité la plus basse et -20 à la priorité la plus haute.

Si le processus est déjà chargé en mémoire, il reste possible de modifier la valeur de "nice" avec la commande **renice.** Le tableau 3 vous donne quelques exemples.

Commande	**Action**
$ **nice -19 find / -name pdf**	lance la commande find avec priorité la plus Basse
$ **renice --20 2332**	place le processus dont le PID est 2332 en priorité la plus haute (seul root peut le faire)

TAB. 3 - Exemples des commandes nice et renice

Vous pouvez visualiser l'évolution du taux d'occupation dans la CPU des processus en temps réél avec la commande top. Une autre commande utile est la commande time qui permet, en la plaçant devant une commande, de vous indiquer les temps d'exécution de celle-ci. Uois temps vous sont délivrés :

- le temps réel passé entre le début et la fin d'exécution (real); - le temps passé en mode utilisateur (user); - le temps passé en mode noyaux (sys).

Exercices

Compréhension générale

Tapez la commande "exec sleep 2". Que se passe t'il et pourquoi ? Affichez ensuite tous les processus vous appartenant. Lancez une commande "sleep 600". Expliquez ce qui se passe

- pour le processus père (le shell) ? - pour le processus "sleep" ?

Processus en tâche de fond

Si le processus "sleep 600" précédent n'est pas encore terminé, comment faites vous pour reprendre la main dans le shell et placer le processus "sleep" en background ? Quelle commande aurait il fallu taper pour obtenir directement ce résultat ? tuez ensuite le processus "sleep".

Redirection de sortie et code de retour

*Comment stocker dans un fichier les résultats donnés par la commande "**find / -size +500k** ? Que s'affiche t'il à l'écran ? Comment stocker ces lignes affichées dans un second fichier ?*

Comment expliquez vous que le code retour de la commande find soit égal à 1 ?

Détournement de signaux

Trouvez une commande qui permette de lancer une commande ps lorsque vous tapez CTRL-C dans votre shell courant. Comment faites vous pour lancer la commande ps à partir d'un autre terminal ?

Détachement

Lancez une commande "sleep 600" de manière à ce que son exécution continue après votre déconnexion. Que constatez-vous lorque vous vous reconnectez ?

Rappels de cours et Travaux Pratiques Linux N°3/4
Première partie : Le Shell: les tubes, les filtres, écriture de scripts

1 Définition

1.1 Les tubes

Le symbole **|,** appelé encore **"tube" ou "pipe",** est utilisé pour relier 2 commandes entre elles. La sortie standard de la commande à gauche du symbole sera utilisée comme entrée standard de la commande de droite comme le montre la figure 1.

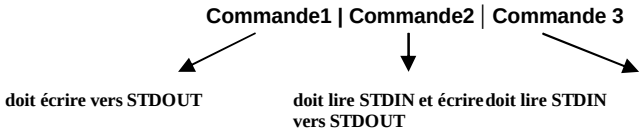

FIG. 1 - Utilisation des tubes

Les redirections d'entrée/sortie semblent similaires à l'emploi des tubes. Pourtant, une différence d'importance existe. Les redirections créent un canal de communication entre un fichier et un processus alors qu'un tube crée ce canal entre deux processus.

1.2 Les filtres

Certains processus peuvent utiliser à la fois l'entrée **STDIN** et les sorties **STDOUT et STDERR.** Un processus qui lit des données sur l'entrée standard et produit des données sur la sortie standard est appelé **filtre.** Vous connaissez le programme **more** qui permet d'afficher le contenu d'un fichier page par page. Cette commande est un filtre et vous pourrez l'utiliser comme une simple commande ou l'associer à d'autres commandes en utilisant un tube. Deux exemples de l'emploi de **more** vous sont donnés dans le tableau 1

Commande	Action
more profile	utilisation de more comme une commande simple
cat profile | more	le résultat de la commande cat est dirigé dans la commande more qui l'affiche à l'écran page par page.

TAB. 1 - Exemples d'utilisation de more

2 les commandes filtres courants

2.1 La commande sort

La commande **sort** permet d'effectuer des tris sur des lignes de texte dans l'ordre numérique (-n), lexicographique (par défaut) ou selon le dictionnaire (-d). Les champs sont délimités par défaut par le caxactère de tabulation mais il est possible de spécifier un autre caractère avec l'option "-t". Il est également possible de trier sur un champs particulier avec l'emploi de l'option -k. L'option "-r" permet d'inverser l'ordre de tri. Le tableau 2 donnent quelques exemples.

Commande	Action
sort -n < /etc/passwd	tri le fichier /etc/passwd par ordre numérique
sort -nt : **-k 3 < /etc/passwd**	tri le fichier passwd par ordre numérique sur le 3ème champ avec : comme délimiteur de champs
sort nrt : **-k 3 < /etc/passwd**	même type de tri en présentant les résultats in-versés

TAB. 2 - Exemples d'utilisation de sort

2.2 La commande grep

La commande **grep** permet la recherche dans des fichiers d'une expression particulière. Les options basiques sont -n qui permet d'afficher les numéros de ligne, -i qui permet de ne pas tenir compte des majuscules et minuscules et -v qui affiche les lignes ne contenant pas l'expression.

Commande	**Action**
grep -i "home" < /etc/passwd	affiche les lignes contenant *home* sans tenir compte des majuscules et minuscules
grep -v "home" < /etc/passwd	affiche les lignes ne contenant pas *home*

TAB. 3 - Exemples d'utilisation de grep

2.3 La commande wc

La commande wc permet de compter le nombre de lignes, de mots et de caractères dans un fichier. Parmi les options, il y a -1 qui affiche le nombre de lignes, -w qui affiche le nombre de mots et -c qui affiche le nombre de caractères. Le tableau 4 vous donne deux exemples d'utilisation de cette commande.

Commande	**Action**
$ **wc –l < /etc/passwd**	compte le nombre de lignes dans */etc/passwd*
$ **wc -c < /etc/passwd**	compte le nombre de caractère dans */etc/passwd*

TAB. 4 - Exemples d'utilisation de wc

2.4 La commande cut

Cette commande extrait des colonnes (option -c) ou des champs (option -f) des lignes d'un fichier ou de l'entrée standaxd. Dans le cas de l'option -f, il est possible de lui spécifier le délimiteur à chercher en utilisant l'option -d. Le délimiteur par défaut est la tabulation. Quelques exemples figurent dans le tableau 5.

Commande	**Action**
$ **cut -f3,7 -d** : **/etc/passwd**	**Filtre** les champs 3 et 7 de chaque ligne de *passwd* En considérant le caractère : comme délimiteur
$ **date | cut -cl-3**	Filtre les caractères 1 à 3

TAB. 5 - Exemples d'utilisation de cut

2.5 La commande head

Cette commande permet d'éditer le début d'un fichier (ou de l'entrée standard) en spécifiant le nombre de lignes (option -n) ou le nombre de caractères (option -c) souhaités.

Commande	**Action**
$ **head -c 1000 /etc/passwd**	Edite à l'écran les 1000 premiers caractères du fi-Chier
$ **head -n 10 /etc/passwd**	Edite les 10 premières lignes du fichier

TAB. 6 - Exemples d'utilisation de head

2.6 La commande tail

Cette commande permet de donner la fin d'un fichier ou de l'entrée standart. Comme avec la commande head, il est possible de spécifier un nombre de caractères (option -c) ou de lignes (option -n). Il est aussi possible de donner un nombre de blocs (512 octets) avec l'option -b.

Commande	**Action**
$ **tail -c 15 /etc/passwd**	Edite les 15 derniers caractères de */etc/passwd*
$ **tail -n 5 /etc/passwd**	Edite les 5 dernières lignes de */etc/passwd*
$ **tail +5 -n /etc/passwd**	**Edite** la fin de */etc/passwd* à partir de la 5ème ligne

TAB. 7 - Exemples d'utilisation de tail

2.7 La commande tr

Cette commande permet de substituer ou de supprimer des éléments pris dans l'entrée standard. Deux chaînes de caractères sont données en argument et précisent les substitutions ou les suppressions à effectuer. La première chaîne de caractères concerne les caractères recherchés alors que la seconde donne les éléments qui viendront en remplacement.

Trois options sont principalement utilisées. Pour les substitutions, l'option -c inverse la recherche alors que l'option -s indique qu'une seule occurrence sera traitée. La suppression est activée par l'option -d.

Commande	**Action**
$ **cat /etc/passwd | tr : "\t"**	remplace les caractères : par une tabulation
$ **cat /etc/passwd tr -d [A-Z]**	supprime tous les caractères majuscule de A à Z
$ **last ▏ tr [:lower :][:upper :]**	remplace toutes les minuscules par des majuscules

TAB. 8 - Exemples d'utilisation de tr

2.8 La commande tee

La commande tee permet une dérivation à l'intérieur d'un tube vers un fichier. Par exemple, si vous souhaitez obtenir un fichier "f1" contenant la liste de votre répertoire et un autre fichier "f2" contenant cette même liste triée, vous taperez :

ls ▏ tee fl ▏ sort > f2

La sortie de la commande ls sera copiée dans fl et dirigée dans la commande **sort.** L'option -a permet d'écrire dans le fichier en concaténant (mode append).

3 Les scripts shell

3.1 Introduction

Dans les précédents travaux pratiques, les différentes commandes ont été exécutées sur la ligne de commande. Il est possible de créer des fichiers contenant un ensemble d'instructions et de réaliser ainsi de véritables programmes - les scripts shell.

3.2 Code retour d'un shell

Le code retour d'un shell est généré par la commande **exit** suivi de la valeur de retour. Cette valeur peut ensuite être visualisée dans le shell par la variable $?.

3.3 Passage d'arguments

Un script shell peut prendre des arguments sur sa ligne de commande. Ces paramètres sont accessibles par des variables internes : la vaxiable $0 représente le nom de votre script, $1 le premier argument, $2 le second... La commande **shift** permet de décaler la numérotation de ces variables. Le nombre des paramètres peut être obtenu grâce à $# alors que $* permet d'obtenir l'ensemble des paramètres.

3.4 Les fonctions

Afin de pouvoir réutiliser le code dans vos script shell, il est possible de définir des fonctions. Celles-ci se définissent grâce au mot réservé **function**

```
function nom-fonction {commande;} ou nom-fonctiono {commande;}
```

Une fonction traite des arguments en respectant les mêmes principes que le passage de paramètres d'un script shell. Le code retour d'une fonction est retourné par le mot clef **return** suivi de la valeur retournée.

Exercices

Utilisation des arguments de la lignes de commande

Créez un script shell qui vous affiche les informations sur la ligne de commande. Le programme affichera ce qui suit

```
$ ./exo1 arg1 22 3

################################################ Information sur la ligne de commande

La ligne de commande comporte 3 paramètres

Execution de la commande: ./exo1

Avec les paramètres: argl 22 3

################################################
```

Informations sur la connexion d'un utilisateur

Complétez le script précédent afin qu'il prenne votre nom de login en paramètre et qu'il affiche grâce à 2 fonctions :

- le nombre de vos connexion sur le système;
- les dates des 3 dernières connexions.

L'affichage généré sera le suivant :

```
################################################ Information sur la ligne de commande

La ligne de commande comporte 1 paramètres

Execution de la commande: ./exol

Avec les paramètres: boebion

################################################ L'utilisateur boebion s'est connecté 100 fois
sur serveur1

Ces 3 dernières connexions ont été aux dates suivantes: Sun Nov 25 18:17 Sun Nov 25 17:25 Sun
Nov 25 14:32
```

Vous obtiendrez les informations sur les connexions grâce aux commandes *last* et *hostname.*

Deuxième partie : Le Shell: programmation et écriture de scripts

1 Les expressions conditionnelles et les structures de contrôle

1.1 La commande test

La commande interne test ou permet de contrôler l'exécution de votre script shell à partir du code de retour d'une expression. Comme nous l'avons déjà vu, un code retour égal à 0 est considéré comme **vrai** alors qu'un code retour non nul sera considéré comme **faux.** Il est alors très simple de tester la bonne exécution d'une commande

```
if [`mv toto titi 2> /dev/null`  ]
then
                echo "Fichier renomme"
else
                echo "Impossible de renommer"
fi
```

L'évaluation d'expressions concernant le type ou les droits d'accès des fichiers sont prévues avec l'utilisation d'options. Les tableaux 1 et 2 vous donnent quelques options.

Options	Vraie si
-a *fichier ou* –e *fichier*	le fichier existe
-b fichier	le fichier existe et est de type bloc
-c *fichier*	le fichier existe et est de type caractère
-d fichier	le fichier existe et c'est un répertoire
-f *fichier*	le fichier existe et c'est un fichier régulier
-p fichier	le fichier existe et c'est un tube nommé
-s *fichier*	le fichier existe et a une taille non nulle
-L fichier	le fichier existe et c'est un lien symbolique
fichierl –nt *fichier2*	*fichierl* est plus récent que *fichier2*
fichierl –ot *fichier2*	*fichierl* est moins récent que *fichier2*
fichierl -ef *fichier2*	les 2 fichiers ont le même numéro d'inode (lien simple)

TAB. 1 - Tests sur le type de fichier

Options	Vraie si
-r *fichier*	le fichier est lisible
-w *fichier*	le fichier est modifiable
-x *fichier*	le fichier est exécutable
-g *fichier*	le fichier possède le bit set-gid
-k fichier	le fichier possède le bit sticky
-u fichier	le fichier possède le bit set-uid
-0 fichier	le fichier a le propriétaire du processus comme pro-
-G fichier	le fichier a le groupe du processus comme groupe

TAB. 2 - Tests sur les droits de fichier

D'autres options permettent de manipuler des chaînes de caractères pour des comparaisons lexicographiques (options =!= < et > et des comparaisons numériques (options -eq, -ne, -1t, -le, -gt et -ge). Vous trouverez celles-ci dans le tableau 3 ci dessous.

Options	Vraie si
-n *chaine*	la chaine est de longueur non nulle
-z *chaine*	la chaine est de longueur nulle
chainel -eq *chaine2*	les 2 valeurs sont égales
chainel -ne *chaine2*	les 2 valeurs sont différentes
chainel -gt *chaine2*	la valeur *chainel* est la plus grande
chainel -ge *chaine2*	la valeur *chainel* est plus grande ou égale
chainel -It *chaine2*	la valeur *chainel* est plus petite
chainel -le *chaine2*	la valeur *chainel* est plus petite ou égale
Chainel = chaine2	les 2 chaines sont égales
chainel != chaine2	les 2 chaines sont différentes
chainel > chaine2	*chainel* est la plus grande
chainel < chaine2	*chainel* est la plus petite

TAB. 3 - Tests sur les chaînes de caractères

Il est possible de combiner ces expressions entres elles avec des parenthèses en utilisant l'opérateur de négation " !" et les opérateurs logiques ET " &&" et OU " 11**".**

1.2 Les structures de contrôle

Vous trouverez ci-dessous un exemple pour chacune des structures de contrôle utilisables dans le bash.

Label1

1.2.1 La construction if

```
if [ -e UNIX5.tex -a -r UNIX5.tex ]
then
       echo -e "Le fichier existe\n"
```

```
	echo -e "Et il est lisible\n"
else
	echo -e "Le fichier n'existe pas\n"
fi
```

1.2.2 La construction case

```
case 'echo $SHELL' in `/bin/bash`) echo -e "Votre shell est le bash\n";; '/bin/tcsh') echo -e
"Votre shell est le tcsh\n";; '/bin/csh') echo -e "Votre shell est le csh\n";; *) echo -e "Je
ne connais pas votre shell\n";; esac
```

1.2.3 La construction for

```
set a b c
for i in   $*
do
		echo $i
done
```

1.2.4 La construction select

La commande select est particulièrement bien adaptée pour réaliser des menus interactifs. En voici un exemple :

```
PS3="Votre Choix: "
select COM in fin ls 'cat /etc/profile'
do
	if [ "$COM" = fin ]
	then
		echo "Vous avez choisi $REPLY"
		exit 0
	else
		echo -e "Vous avez choisi $REPLY\n" $COM
	fi
done
```

1.2.5 La construction while

```
let I=0
while [ $I -lt 10 ]
do
	echo $I
	let I=I+1
done
```

1.2.6 La construction until

```
let I=0
until [ $I -gt 256 ]
do
	echo $I
	let I=I+2
done
```

1.2.7 le mot clef break

Il permet de forcer la sortie d'une boucle **for, while ou until.** Le mot clef **break** peut prendre un argument strictement supérieur à 1 afin de spécifier le nombre de boucles imbriquées dont il faut sortir. Si la valeur spécifiée est supérieure au nombre de boucles imbriquées, on sort de toutes les boucles imbriquées.

1.2.8 le mot clef continue

Il permet de passer à la prochaine itération d'une boucle **for, while ou until.** Le mot clef **continue** peut aussi prendre un argument strictement supérieur à 1 afin de spécifier le nombre d'itérations à "sauter". Si la valeur spécifiée est supérieure au nombre d'itérations possibles, la dernière itération est toutefois exécutée.

1.2.9 Le mot clef exit

Il permet de stopper le programme en précisant en argument le code retour. Par convention, ce code retour est un entier nul en cas de sortie normale du programme et non nul en cas de fin de programme avec erreur.

2 Saisie sur la ligne de commande

Afin de réaliser un script shell interactif, il est nécessaire de pouvoir saisir des valeurs. La commande interne au bash **read** interrompt le programme pour demander une saisie terminée par un retour chariot. La commande peut être suivie d'une variable pour stocker la valeur.

```
$ read A
$ jjd djdjd
$ echo $A
$ jjd djdjd
```

3 Calcul numérique

La commande interne **let** permet d'effectuer des calculs arithmétiques dans le shell. Cependant l'interpréteur de commande bash ne permet pas d'effectuer des calculs en virgule flottante. Pour ces opérations, nous sommes obligés de faire appel à une commande externe, la commande **bc**. Cette commande prend des données à calculer sur son entrée standard et affiche le résultat sur sa sortie standard. L'option "-l" permet de charger la bibliothèque mathématique.

$ let I=23
$ let J=I+100
$ echo $J
$ 123
$ `RES=`echo ¾ | bc –lc``
$ echo $RES
$ .75000000000000000000

4 Gestion évolué de la ligne de commande

La syntaxe générale " commande *[-options] arguments"* est utilisée pour les commandes Unix. Nous avons étudié dans un précédent T.P. comment obtenir les argument de la ligne de commande ($0, $1, ...). Il existe une commande qui permet une gestion plus évoluée des arguments de la ligne de commande - **getopts.** Prenons l'exemple suivant

```
while getopts ":ab:c" opt
do
        case $opt in
                a)      echo "Traitement de l'option -a";;
                b)      echo "Traitement de l'option –b"
                        echo "L'argument est $OPTARG";;
                c)      echo "Traitement de l'option –c
                ?)      echo "Usage: commande [-a] [-b arg] [-c] arguments"
                        exit 1;;
        esac
done
shift $(($OPTIND-1))
echo "$1 est le premier argument après les options ... "
```

L'appel à getopts dans la boucle while prépare la boucle à accepter les options -a, -b et -c et spécifie que l'option -b doit traiter un argument (:). Le : qui se trouve au début permet de traiter la saisie d'options invalides et place ? dans la variable $opt. Si une option prend un argument, la valeur de celui-ci sera contenu dans **$OPTARG**

Pour chaque option spécifiée sur la ligne de commande, une variable nommée **$OPTIND** est incrémentée. Par exemple, si vous spécifiez 2 options (-ab), la variable vaudra 2. Cela explique le décalage des arguments qui est effectué par la commande **shift** afin de retrouver les "vrais" arguments de la commande avec les variables $1, $2, etc...

Exercices

Conversion Euros/Francs et Francs/Euros

Première partie

Réalisez un script appelé "Convertisseur" qui vous permettent de convertir en Euros une somme en Francs passée en argument. Le taux de conversion sera contenu dans un fichier nommé Taux sous la forme : Taux :6.55957. Pensez à tester si ce fichier est existant et lisible par votre processus. Si ce n'est pas le cas, votre script doit afficher une erreur et quitter en indiquant un code retour 1.

Seconde partie

Modifiez votre script afin de pouvoir traiter l'option -e qui permet d'inverser la conversion (Euros vers Francs). En cas d'erreur de syntaxe, le script affiche l'usage du programme et quitte avec un code retour égal à 2.

Dernière partie

Modifiez votre script afin de pouvoir traiter l'option -f FILE qui permet d'indiquer un fichier dans lequel se trouve plusieurs sommes à convertir (une par ligne). Les résultats des conversions seront placés dans un fichier nommé "Resultats".

Rappels de cours et Travaux Pratiques Linux N°4/4
Programmation C sous linux

1 Le langage C

Introduire un langage de développement dans des travaux pratiques sur un système d'exploitation peut vous paraître curieux. Si vous devez travailler sur des systèmes utilisant UNIX ou Linux, vous constaterez que le langage C est incontournable. Le noyau, les différents interpréteurs de commande (csh, bash ...), les commandes du système d'exploitation (Is, cp, rm ...) sont écrits avec le langage C. Si vous voulez enrichir le système avec une application nouvelle, celle-ci sera le plus souvent écrite avec ce langage et vous devrez parfois savoir la compiler.

Le **langage C** est un langage d'intérêt général et il est portable. Il n'est pas affecté à un type de programmation comme, par exemple, le **Cobol** pour les bases de donnée ou encore le **Fortran** pour le calcul. Il offre la possibilité d'effectuer des opérations de bas niveaux, opérations qui permettent l'écriture de systèmes d'exploitation. Il possède un jeu d'instructions limité enrichie par de nombreuses librairies. La grande majorité des systèmes d'exploitation ont un compilateur C et un effort de normalisation a été effectué à travers la norme **ANSI.**

Le compilateur Gnu C est intégré dans toutes les distributions Linux. C'est avec celui-ci que nous allons travailler.

2 Construction d'un programme C

2.1 Déroulement d'une compilation

L'écriture d'un programme C donne lieu à la création d'un ou plusieurs fichiers sources (extension c) contenant les éléments du langage. La création de fichiers **header** personnels (extension h) permet de définir les fonctions créées par l'utilisateur. Une fois ces fichiers prêts, l'appel aux compilateur peut se faire. Avec Gnu C, la ligne de commande minimale sera

gcc source.c

Cette ligne de commande crée un fichier exécutable **a.out.** Pour nommer précisément le binaire, l'option -o du compilateur suivi du nom est utilisée

gcc -o binaire source.c

La création d'un fichier exécutable à partir de fichiers sources en C fait appel à plusieurs programmes comme le montre la figure 1.

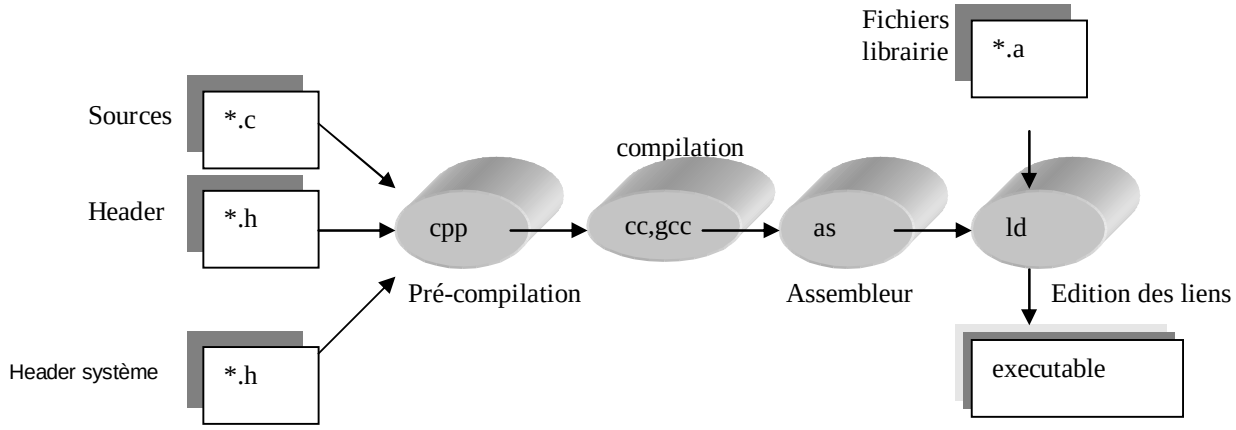

FIG. 1 - Déroulement d'une compilation

Le précompilateur (cpp) est d'abord lancé. Il supprime les commentaires et interprète les directives de compilations qui se trouvent dans les fichiers header ou sources.

L'appel au compilateur (cc ou gcc) permet en une ou deux passes de générer du code en assembleur.

L'assembleur (as) crée un fichier objet correspondant au fichier source qui sera traité ensuite par l'éditeur de lien (ld). Cette dernière opération produit l'exécutable final en réunissant les fichiers objets de l'utilisateur et tous les fichiers objet des librairies nécessaires.

2.2 Les options du compilateur

Nous venons de voir une première option (-o) du compilateur. Il en existe de nombreuses (voir le manuel en ligne). Le tableau 1 vous présente quelques unes de ces options.

Options	Caractéristiques
-V	Visualise toutes les actions effectuées lors d'une compilation.
-C	Supprime l'édition de lien. Seul le(s) fichier(s) o est (sont) généré(s).
-0	Spécifie le nom de l'exécutable à générer.
-S	Ne génère pas le fichier exécutable et crée un fichier s contenant le programme en assembleur.
-L	Indique un chemin de recherche supplémentaire à l'éditeur de liens pour d'autres Librairies.
-l	Indique une nouvelle librairie à charger.
-I	Indique un chemin de recherche supplémentaire pour des fichiers de définition (.h)

TAB. 1 - Quelques options du compilateur C

2.3 Exercice

Créez le fichier source **bonjour.c** qui contient les instructions C suivantes

```
# include <stdio.h>

int main(void)
{
     printf("bonjour\n");
    exit(0);
}
```

Compilez-le pour créer l'exécutable **bonjour** et testez-le. Recompilez ce programme en spécifiant l'option -v et commentez ce qui s'affiche.

Trouvez l'option pour ne générer que le fichier assembleur puis objet de votre source.

3 Construction de librairie

Il est intéressant de se construire des librairies contenant les fonctions les plus fréquemment utilisées plutôt que de les réécrire à chaque projet. Il suffit ensuite d'indiquer vos librairies au moment de la compilation. Pour cela, les options -L et -1 permettent respectivement d'inclure un nouveau chemin de recherche pour l'éditeur de lien et d'indiquer le nom de librairie.

3.1 Exercice : Création d'une librairie

Vous trouverez ci-dessous 2 fonctions qui affichent "Bonjour" et "Au revoir" autant de fois que le nombre passé en argument

```
void bonjour(int nbre)
{
      int i;
      for(i=0;i<nbre;i++) printf("Bonjour ... \n");
      return;
}
void revoir(int nbre)
{
      int i;
      for(i=0;i<nbre;i++) printf("Au revoir ... \n");
      return;
}
```

Placez ces fonctions dans des fichiers nommés **bonjour.c et revoir.c.** Compilez ces fichiers afin de générer les fichiers objet :

gcc -c bonjour.c revoir.c

Créez la librairie **libsalut.a** à l'aide de la commande ar : **ar r libsalut.a bonjour.o revoir.o**

Vous pouvez vérifier le contenu de votre librairie en tapant **ar t libsalut.a**

3.2 Exercice : Utilisation de la librairie

A partir de cette étape, vous pouvez déjà utiliser ces fonctions en respectant quelques conditions. La première de celles-ci est de placer **libsalut.a** dans le répertoire où vous effectuez votre compilation. La seconde est de compiler votre programme en spécifiant le nom de la librairie. Tapez le programme suivant dans le fichier **exemple2.c**

```
#include <stdio.h>


int main(int argc,char **argv)
{
      int a ;

      if(argc != 2)
      {
            printf("usage :%s nombre\n",argv[0]) ;
            exit(1) ;
      }

      a=atoi(argv[1]);

      printf("\n") ;

      bonjour(a);
      revoir(--a);
      exit(0);
}
```

Compilez ce programme en exécutant la ligne suivante

gcc -o exemple exemple.c libsalut.a

Vous pouvez ensuite placer toutes vos librairies dans un de vos répertoires (par exemple, ~/librairies). Vous lancerez alors votre compilation de la manière suivante

gcc -o exemple exemple.c –L~/librairies -lsalut

4 Compilation séparée

4.1 Présentation

Lorsque vous réalisez de gros projets, vous devez découper ceux-ci en plusieurs programmes sources. L'avantage est un gain de temps au moment de la compilation. Si une application est constituée des 3 fichiers sources **sourcel.c, source2.c et source3.c** la compilation d'un exécutable nommé **exec** est effectuée par la ligne suivante :

gcc -o exec sourel.c source2.c source3.c

Nous connaissons l'option -c du compilateur qui permet de générer des fichiers objets sans effectuer une édition de liens. Si vous modifiez un des fichiers source (par exemple, **source3.c),** nous utilisons cette option pour ne compiler que le fichier modifié

gcc -c source3.c
gcc -o exec source3.o sourcel.o source2.o

5 Les fichiers make

5.1 Intérêt et syntaxe

Lorsqu'on réalise un projet en le découpant en plusieurs modules, vous constatez qu'il est difficile de se rappeler ce qu'il faut recompiler après une modification.

La commande **make** permet de maintenir un programme automatiquement. Pour cela, elle utilisera un fichier de description nommé en général **makefile ou Makefile** qui contient des dépendances et les actions à mettre en oeuvre pour maintenir le programme.

La syntaxe de la commande make est la suivante .

make [-f nom-fichier] [-arg-optionnels] [ref..]

L'option -f indique le nom du fichier de description. En général, la commande est lancée sans argument. Dans ce cas, elle recherche un fichier de description sur le répertoire courant.

Parmi les arguments optionnels, vous trouverez :

- -s (mode silencieux) : pas d'affichage des commandes avant exécution; - -n : l'ensemble des commandes à effectuer est affiché mais non exécuté; - -d (mode debug) : le maximum d'informations est affiché.

5.2 Contenu d'un fichier Makefile

Un fichier **Makefile** contient :

- les dépendances entre les fichiers sources composant votre application; - les actions lancées suivant la modification d'un de ces fichiers sources; - des variables qui permettent de fixer des chemins, des noms d'exécutable etc...

5.3 Description du fichier Makefile

On écrit un fichier **Makefile** en plaçant d'abord les variables générales. Les dépendances de l'exécutable suivent ensuite avec l'action associée. Si ce n'est pas le cas, vous devrez spécifier le nom du programme à générer comme paramètre de la commande **make.** On termine enfin le fichier en déclarant toutes les autres dépendances.

Il est conseillé d'utiliser la tabulation lorsqu'on sépare un champs d'un autre. La commande make gère difficilement les espaces. Le fichier **Makefile** suivant permet de maintenir le projet **prog** :

```
OBJS=menu.o fichier.o date.o
```

```
EXEC=prog
INSTALLDIR=.
LIBS=~/librairie
CC=gcc

prog: $(OBJS)
      $(CC) $(OBJS) -o $(EXEC)
date.o: date.c header.h
      $(CC) -c date.c
menu.o: menu.c header.h
       $(CC) -c menu.c
fichier.o: fichier.c header.h
      $(CC) -c fichier.c
install:
      cp $(EXEC) $(INSTALLDIR)/prog2
clean:
      rm *.o prog
```

EXEC est une variable contenant le nom de l'exécutable à générer. **OBJS** est une variable contenant tous les fichiers objets nécessaires à la construction de **prog.** Ce qui suit est la liste des dépendances et les actions associées. Le ou les fichiers qui dépendent d'autres fichiers sont précédés du caractère :. On peut interpréter de la manière suivante :

- le fichier **prog** doit être compilé si on modifie **date.c, menu.c** ou **fichier.c.** On peut indiquer directement les fichiers objets **.o** car la commande make sait de manière implicite qu'un fichier **.o** dépend directement d'un fichier **.c**. La modification d'un fichier **.c** lancera donc automatiquement sa compilation en **.o** et la compilation de **prog** par la commande qui suit cette ligne.
- les fichiers **menu.o, fichier.o**, et **date.o** dépendent d'une modification de **header.h**. Si ce dernier est modifié, une compilation complète pour générer **prog** sera effectuée.
- Les lignes **clean et install** ne possèdent pas de dépendances. Seules des actions sont définies. En faisant **make clean,** vous supprimerez les fichiers **prog et .o**.

5.4 Exercices

Créez ce fichier **Makefile.** Supprimez tous les fichiers objets de votre répertoire et lancez la commande **make.** Modifiez les différents fichiers sources et relancez de nouveau **make.**

Exercices à rendre : L'écriture de scripts

Exercice 1

Ecrire une procédure **choix** qui demande « Voulez-vous continuer ? », puis saisit un choix qui sera stocké dans la variable réponse. A partir de choix, si la saisie est du format d'un des modèles (N ou n ou no ou NO) alors on sortira de la procédure avec un code de retour 0.

Exercice 2

Ecrire un script **existfic** qui permet de déterminer s'il s'agit d'un répertoire ou d'un fichier ordinaire et qui autorise l'écriture, la lecture, ou l'exécution.
Par exemple, le premier argument de la procédure sera le nom du fichier et le second le droit d'accès voulu.
Exemple : exisfic toto r

toto est un fichier ordinaire accessible en lecture

Exercice 3

Ecrire une procédure **engtofr** qui permet la traduction des chiffres anglais (one, two, three, four, ..., nine) en français.

Exercice 4

A l'aide de la commande date d'UNIX, réalisez votre shellscrip permettant d'obtenir la date en français.

Exercice 5

Créez un fichier nommé fruits qui contient la liste suivante :

Bananes
jaunes
Oranges
oranges
Pommes
rouges
Raisins
noirs

Ecrire un script qui affiche uniquement les fruits ou uniquement les couleurs selon le paramètre fourni (1 pour les fruits, 2 pour les couleurs).

Exercice 6

Que calcule la fonction f ainsi définie:

```
f(x,n)
int x,n;
{
        int r=1, x2=x,masque=1;
        do
        { if(n&masque)
        r *=x2;
        x2 *=x2;
        }while(n >>=1);
return r;
}
```

déroulez l'exécution sur un exemple. Incorporez cette fonction dans votre bibliothèque de fonction mathématiques et montrez son utilisation dans un main.