Cours utilisateur UNIX

Olivier Hoarau

V1.0, 27 décembre 1999

Ce document est un essai de vulgarisation d'UNIX. Il est destiné à ceux qui l'utilisent déjà mais aussi aux débutants qui voudraient en apprendre plus sur ce système d'exploitation qui est en train de se populariser à vitesse grand V avec l'explosion de Linux. La dernière version de ce document est consultable sur le site officiel http://www.linux-france.org/article/ohoarau. La version en construction se trouve sur le site http://funix.free.fr. Toutes remarques et suggestions concernant ce document sont à expédier à Olivier Hoarau mailto:olivier.hoarau@fnac.net.

6. Commandes UNIX et redirection

7. Le système de fichiers

8. Les droits d'accès

9. Gestion des processus

10. Les titres UNIX

11. Les commandes grep et find

12. Expressions régulières et sed

13. La commande awk

---

1. Historique du document

27 décembre 1999 version 1.0 du document.

2. Préambule

UNIX est un système d'exploitation d'une richesse incroyable, il serait bien prétentieux d'essayer en quelques pages d'en faire le tour. C'est pourquoi je me suis fixé comme objectif de ne présenter que les commandes les plus courantes qui permettront à un utilisateur de se débrouiller avec n'importe quel système UNIX, de HP-UX à Solaris en passant par Linux sans oublier les autres.

Ce document s'adresse à toute personne ayant une petite expérience d'un système informatique, il s'adresse aussi à ceux connaissant déjà UNIX qui voudraient approfondir certaines notions.

Le but de ce document n'est pas de traiter de l'administration d'un système UNIX.

3. Principes de base UNIX

3.1 Les processus ou process

Tout logiciel est à la base un programme constitué d'un ensemble de lignes de commandes écrites dans un langage particulier appelé langage de programmation. C'est uniquement quand on exécute le logiciel que le programme va réaliser la tâche pour laquelle il a été écrit, dans ce cas là on dira qu'on a affaire à un processus ou process. En d'autres termes le programme est résolument statique, c'est des lignes de code, alors que le process est dynamique, c'est le programme qui s'exécute.

Par exemple le logiciel Winword sous Windows est en fait un bête programme écrit dans un langage abscons qui a été ensuite compilé pour le rendre compréhensible par la machine, ce n'est uniquement que quand vous le lancez, que vous avez alors affaire au process Winword.

3.2 Définition d'un système d'exploitation

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

Un exemple vaut mieux qu'un grand discours, quand vous ouvrez un fichier dans votre traitement de texte favori, vous avez appuyé sur l'icône qui va bien, votre traitement de texte interprète l'action d'ouverture de fichier et transmet l'ordre au système d'exploitation, ce dernier va alors commander au contrôleur du disque dur de chercher les pistes correspondantes sur le disque qui correspondent au fichier en question. Normalement un logiciel ne devrait jamais " discuter " avec le matériel, le système d'exploitation se place entre les deux pour transmettre et éventuellement rejeter des commandes illicites.

3.3 Environnement

Un environnement est dit fenêtré quand il y a possibilité de pouvoir faire apparaître plusieurs fenêtres, il va de pair avec l'utilisation d'une souris, Windows est par exemple un exemple d'environnement fenêtré. On parle aussi d'environnement graphique.

A l'opposé on trouve aussi des environnements textuels non graphiques, DOS en est un bel exemple.

4. Présentation du système UNIX

4.1 Les utilisateurs UNIX

Sur un système UNIX, on trouve deux types de personnes, celle qui va utiliser le système dans le but de produire quelque chose, le système UNIX est pour elle un moyen, un outil. Cette personne est l'utilisateur UNIX, on peut trouver dans cette catégorie, le programmeur, l'utilisateur de base de données, etc. La deuxième catégorie de personnes est chargé de l'installation, de la configuration et de la bonne marche du système UNIX, ce sont les administrateurs systèmes UNIX.

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

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

4.2 Les fonctions principales

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

Partage des ressources équitables

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

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

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

Interface avec le matériel

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

Gestion de la mémoire

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

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

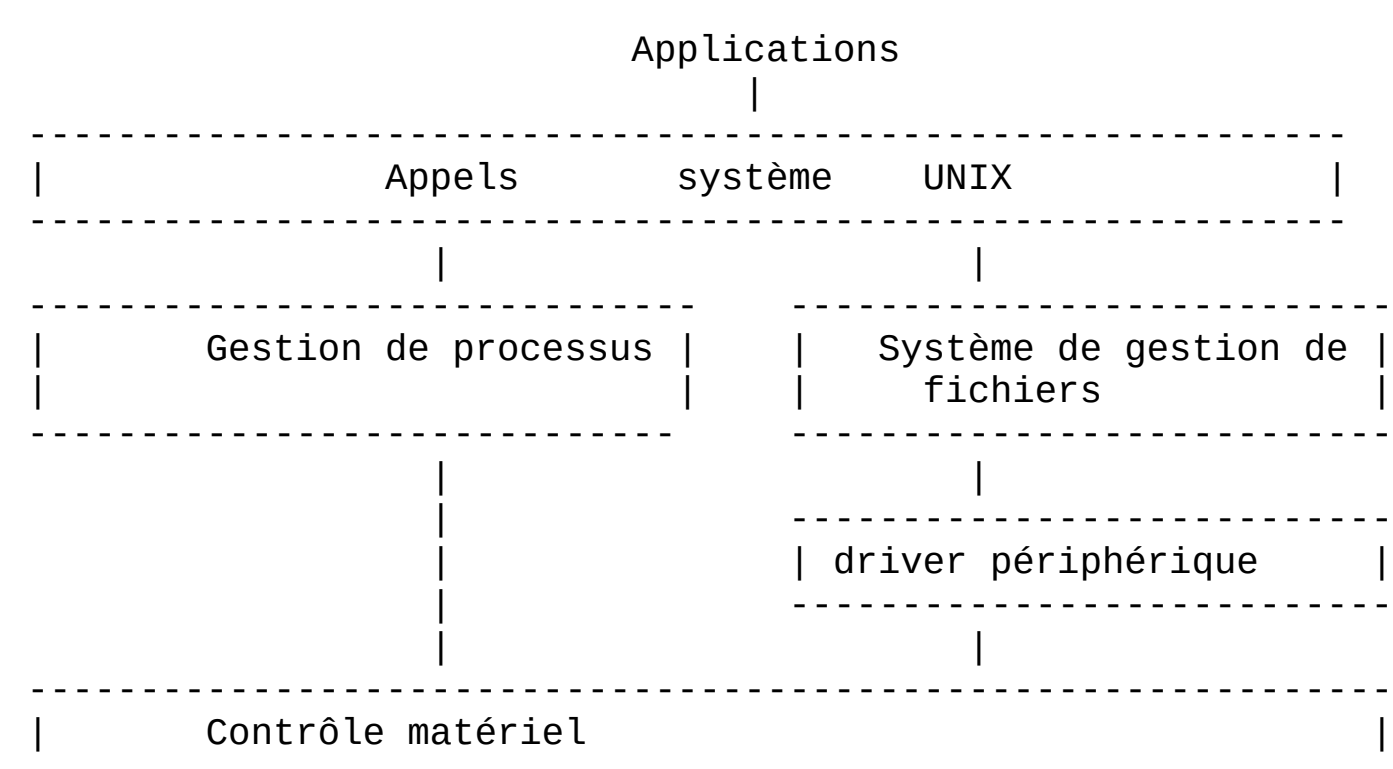
La mémoire est vue comme une ressource matérielle, UNIX doit donc vérifier qu'aucun process accède à la mémoire directement ou ne se réserve une zone de la mémoire.

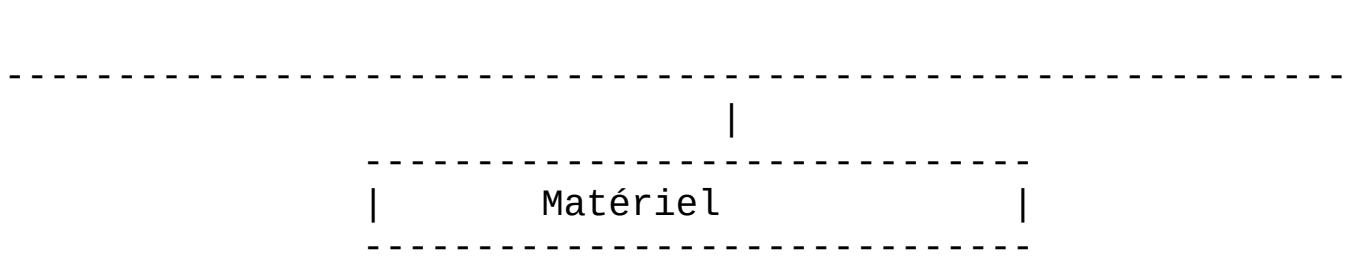
Gestion des fichiers

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

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

4.3 Structure du système UNIX





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

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

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

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

4.4 Le shell

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

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

Next Previous Contents

5. Ouverture et fermeture de session

5.1 Ouverture de session

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

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

A la mise sous tension, vous pouvez aussi disposer d'une interface graphique de connexion, au lieu d'avoir un simple **login:** avec le curseur qui clignote, vous avez une fenêtre ou bannière qui vous invite à saisir votre login et votre mot de passe. C'est notamment le cas pour la configuration par défaut de la Mandrake 6.0, mais aussi pour les versions récentes de HP-UX et de Solaris.

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

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

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

5.2 Changement de password

En vous déclarant sur la machine, on vous a imposé un mot de passe, vous pouvez le changer, pour cela vous disposez de la commande **passwd**. Certains UNIX font en sorte que vous ne puissiez pas saisir un mot de passe simple, il faudra mettre au moins 6 caractères, avec au moins un, voie deux, caractère non alphabétique.

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

ATTENTION : Evitez de vous servir du pavé numérique, car d'un poste à un autre, il peut y avoir des grosses différences à ce niveau là.

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

```
>passwd
Old passwd :******
Setting password for user : olivier
New password :******
Reenter password :******
>
```

ATTENTION : Sur certains systèmes, on ne doit pas taper **passwd** mais **yppasswd**, demandez le à votre administrateur. Pour informations, on utilise **yppasswd** pour les client NIS.

5.3 Fermeture de session

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

Dans un environnement graphique, vous avec une commande Exit, ou Logout, qui a strictement le même effet.

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

Next Previous Contents

6. Commandes UNIX et redirection

6.1 Syntaxe d'une commande

La syntaxe standard d'une commande UNIX est la suivante :

```
commande -options arg1 arg2 arg3
```

Les options varient en fonction de la commande, le nombre des arguments qui suivent dépend aussi de la commande, par exemple la commande :

```
sort -r mon-fichier
```

sort (trier) permet de trier un fichier, l'option r (reverse), permet de trier en sens inverse le fichier. L'argument unique de la commande est le nom du fichier. Avec

```
cp -R mon-repertoire nouveau-repertoire
```

La commande **cp** (copy) copie un répertoire (option R) vers un autre répertoire, on a ici deux arguments.

On peut coupler deux options : **ps -ef**, avec cette commande on a l'option **e** et **f** (voir plus loin la signification de la commande).

A noter que pour introduire une option on met -, ce n'est pas nécessaire pour certaines commandes (tar par exemple).

6.2 Les entrées sorties

Il y a trois sortes d'entrées sorties ou flux de données : le premier est l'entrée standard, c'est à dire ce que vous saisissez au clavier, le deuxième est la sortie standard, c'est à dire l'écran, plus précisément le shell, et le troisième est la sortie standard des messages d'erreurs consécutifs à une commande, qui est généralement l'écran.

Chacun de ces flux de données est identifié par un numéro descripteur, 0 pour l'entrée standard, 1 pour la sortie standard et 2 pour la sortie standard des messages d'erreur.

6.3 Redirection des entrées sorties

Quand vous lancez une commande dans un shell, il peut y avoir du texte qui s'affiche suite à l'exécution de la commande, ce texte par défaut, s'affiche dans le shell. On dit que le shell (ou terminal) est la sortie standard, c'est là où va s'afficher tous les commentaires d'une commande.

Vous pouvez changer ce comportement, en tapant :

```
ma-commande > mon-fichier
```

Tous les commentaires, les sorties, de la commande, ne vont pas apparaître au shell mais être écrits dans un fichier. En d'autres termes, la standard standard est redirigé vers un fichier. Cela peut être utile, si vous avez une commande qui génère énormément de commentaire, et que vous voulez les récupérer, pour les exploiter par la suite, à la terminaison de la commande.

La redirection > a pour effet de créer le fichier **mon-fichier**, si ce fichier existait déjà, il est tout simplement écrasé (supprimé et recréé), ce qui peut être gênant si vous ne voulez pas perdre ce qu'il contient, vous disposez donc de la redirection >>. En tapant :

```
ma-commande >> mon-fichier
```

Le fichier **mon-fichier** n'est pas écrasé, mais la sortie standard (les commentaires de la commande) sont ajoutés en fin de fichier, à la suite du texte qui était déjà dans le fichier.

Les redirections marchent dans les deux sens, par exemple en tapant la commande suivante :

```
sort < mon-fichier
```

Vous envoyez le contenu du fichier **mon-fichier** vers la commande **sort** (trie), celle-ci va donc trier le contenu du fichier, par défaut le résultat sort sur la sortie standard, c'est à dire à l'écran, plus précisément sur le shell. Avec :

```
sort < mon-fichier > fichier-trie
```

On a vu que **sort** < **mon-fichier** avait pour effet de trier le fichier **mon-fichier**, l'expression **>fichier-trie** a pour effet d'envoyer le résultat (le fichier trié) dans un fichier **fichier-trie**, le résultat n'apparaît plus à l'écran, mais est sauvegardé dans un fichier.

Avec la redirection << la commande va lire les caractères jusqu'à la rencontre d'une certaine chaîne de caractères. Exemple avec la commande **cat** (catalogue, permet d'éditer le contenu d'un fichier).

```
>cat << fin je tape du texte jusqu'à la chaîne de caractère fin >
```

En tapant la commande, vous revenez à la ligne, mais perdez le prompt, **cat** va lire (et éditer) les caractères que vous saisissez jusqu'à qu'il rencontre la chaîne fin, à ce moment là, le prompt apparaît à nouveau. Si vous voulez créer un fichier avec un peu de texte à l'intérieur, vous ferez :

```
>cat << fin > mon-fichier je tape du texte qui sera sauvegardé dans mon-fichier, pour terminer le texte fin >
```

Le texte que vous venez de saisir, se trouve donc dans **mon-fichier**.

Avec la commande :

```
>fichier-vide
```

Vous créez un fichier vide **fichier-vide**.

6.4 Redirection des erreurs

Par défaut les messages d'erreur s'affichent à l'écran (sortie standard par défaut), vous pouvez modifier ce comportement. On rappelle que la sortie d'erreur a pour code 2. Vous pouvez sauvegarder dans un fichier vos messages d'erreur, pour analyse ultérieure, en tapant :

```
cat mon-fichier 2>fichier-erreur
```

Si on rencontre une erreur pendant l'exécutionde la commande d'édition **cat** de **mon-fichier** (absence du fichier par exemple), le message d'erreur sera sauvegardé dans le fichier **fichier-erreur**.

En tapant :

```
sort mon-fichier > fichier-trie
```

Vous redirigez le résultat de la commande **sort mon-fichier** vers le fichier**fichier-trie**, la sortie standard (descripteur 1) n'est donc plus l'écran (plus précisément le shell ou terminal) mais le fichier **fichier-trie**.

Par défaut les messages d'erreur s'affichent dans le shell, vous pouvez faire en sorte qu'ils s'affichent dans le fichier **fichier-trie**, en tapant :

```
sort mon-fichier > fichier-trie 2>&1
```

Avec la syntaxe **>&** vous indiquez que les messages d'erreurs seront redirigés vers la sortie standard qui est le fichier **fichier-trie**.

6.5 Les pipes

Un pipe (en français tube de communication) permet de rediriger la sortie d'une commande vers une autre. En d'autres termes, pour rediriger les résultats (la sortie) d'une commande, on a vu qu'on pouvait taper :

```
commande1 > sortie1
```

On redirige cette sortie vers une autre commande, ça devient donc une entrée pour cette dernière commande, pour cela vous tapez :

```
commande2 < sortie1
```

En fait la syntaxe **commande1|commande2** (| étant le symbole de pipe) est totalement équivalente aux deux lignes de commandes précédentes.

Exemple : **ls** permet la visualisation de fichiers, en tapant **ls**, on obtient :

```
fichier1 fichier2 totofichier
```

grep permet la recherche d'une chaîne de caractère dans une liste donnée, en tapant **grep toto *** (***** signifie tous les fichiers, **grep** recherche la chaîne de caractère **toto** dans les noms de tous les fichiers), on obtient :

```
totofichier
```

On a le même résultat avec le |, en tapant :

```
ls | grep toto
```

La première commande aura pour effet de lister le nom des fichiers se trouvant à l'endroit où l'on a tapé la commande, la sortie standard (le résultat de la commande) est donc une liste de nom, elle est redirigée vers la commande **grep**, qui va y chercher une chaîne de caractère contenant **toto**. Le résultat est donc aussi:

```
totofichier
```

---

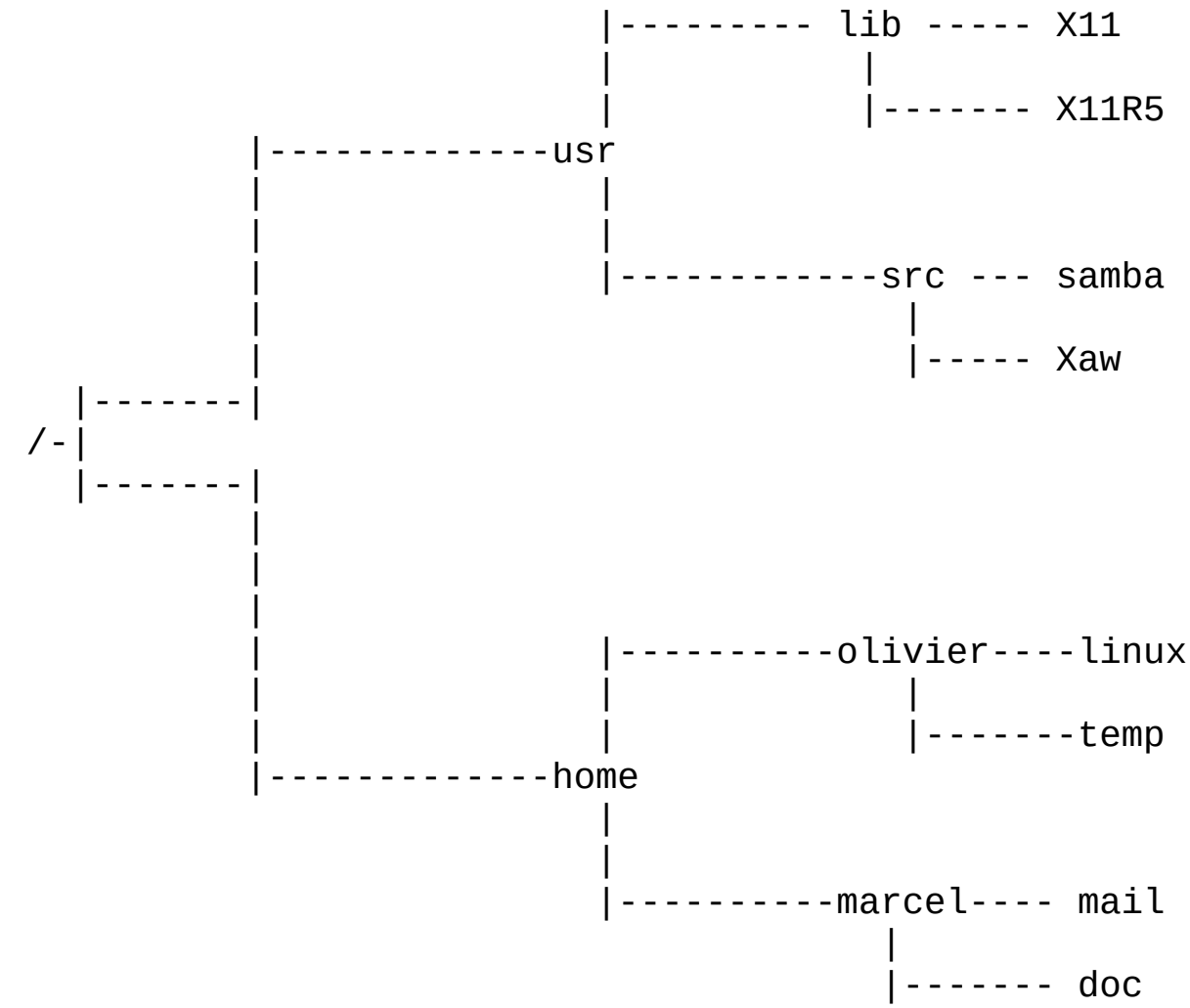
7. Le système de fichiers

7.1 Les types de fichier

Il existe trois types de fichier, le fichier qu'on pourrait qualifié de normal, le répertoire ou catalogue (en anglais directory) et les fichiers spéciaux.

Un fichier normal contient des données, ce fichier peut être lisible, c'est à dire contenir des informations compréhensibles écrites en claire, ce fichier peut être aussi totalement illisible. Concrètement un fichier texte qui comme son nom l'indique contient du texte est lisible, alors qu'un exécutable ne l'est pas, si vous cherchez à l'éditer vous ne verrez rien de compréhensible, dans ce dernier cas, on dit aussi qu'on a affaire à un fichier binaire.

Un répertoire peut être considéré comme un classeur, dans lequel on met des fichiers, c'est un élément d'organisation de l'espace du disque dur. Les fichiers ayant les mêmes " affinités " peuvent ranger sous un même répertoire, de même on peut trouver des sous répertoires dans un répertoire, qui eux mêmes contiennent des fichiers et d'autres sous répertoires. Ce système hiérarchique fait penser à un arbre, d'où le terme d'arborescence.



Il existe un " ancêtre " à tous les répertoires, c'est la racine ou le / (slash) sur le shéma. Tout répertoire, qui n'est pas la racine elle même, possède un répertoire qui le contient (appelé répertoire père) et peut posséder des sous-répertoires (répertoires fils) et des fichiers .

Quand on crée un répertoire, le système crée automatiquement deux " fichiers " sous le répertoire, le premier est un " **.** ", qui représente le répertoire lui-même, le deuxième est un " **..** " qui représente le répertoire père.

Le troisième type de fichier est le fichier dit spécial, qu'on a abordé brièvement auparavant, rappelons que l'on doit passer par eux si on veut dialoguer avec un périphérique matériel.

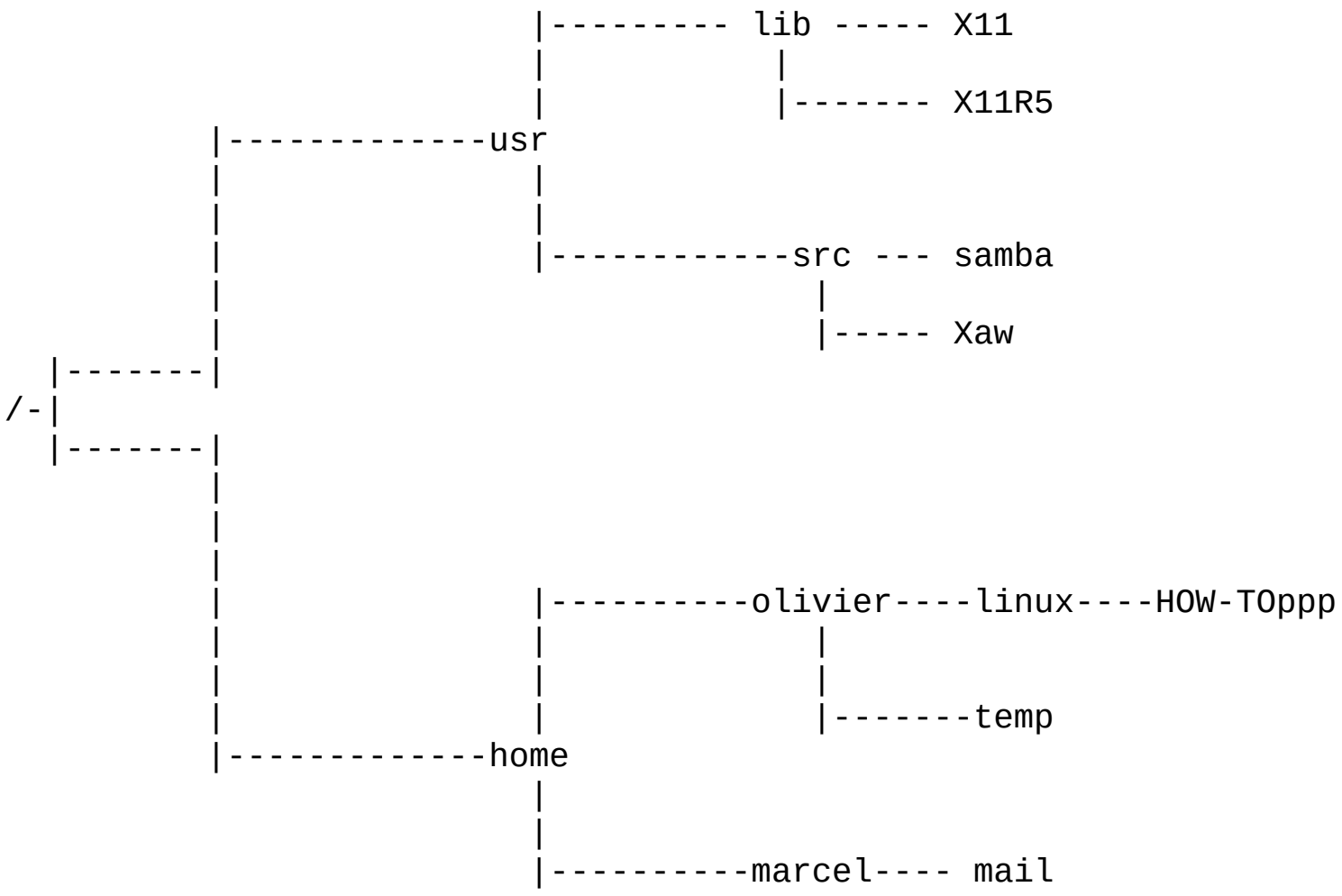
7.2 Atteindre un fichier

Nommer un fichier

Tout fichier, qu'elle que soit son type, doit pouvoir être identifié, c'est pourquoi on les nomme avec un nom en rapport avec le fichier. Ce nom comporte au maximum 255 caractères, sachant qu'il existe une distinction entre les lettres minuscules et majuscules, et que certains caractères sont interdits, ce sont par exemple le /, les parenthèses **()**, l'espace ou *****.

Le chemin d'accès

Ce fichier est rangé dans un répertoire du système de fichiers, on doit pouvoir y accéder, en suivant un chemin dans l'arborescence.



```
                                    |
                                    |------- doc
```

Pour indiquer le chemin du fichier (**HOW-TOppp** dans notre exemple), on part de la racine (/), on indique le premier répertoire traversé, puis les autres, en séparant chacun des répertoires d'un /. Ainsi donc pour notre fichier le chemin d'accès est :

```
/home/olivier/linux
```

En indiquant **/home/olivier/linux/HOW-TOppp** le fichier est parfaitement identifié, en effet on sait où le trouver puisqu'on a son chemin, et le nom du fichier **HOW-TOppp**.

A noter qu'on peut avoir des fichiers portant le même nom dans le système de fichiers dés lors qu'ils n'ont pas le même chemin, et donc qu'ils ne se trouvent pas au même endroit.

On dit que le chemin du fichier est absolu parce qu'à la vue de son chemin d'accès, en partant de la racine, on sait exactement où se trouve le fichier.

Un chemin est dit relatif, quand il n'est pas nécessaire, d'indiquer le chemin complet, de l'endroit où on se trouve dans l'arborescence il suffit de rajouter le chemin par rapport à ce même endroit.

En admettant qu'on se trouve sous **/home/olivier**, si l'on veut accéder à notre fichier **HOW-TOppp**, le chemin relatif au répertoire courant est **./linux**, le point représentant le répertoire courant comme on l'a vu auparavant. Ce qui donne en chemin absolu **/home/olivier/linux**.

Les commandes

La commande pour se déplacer dans l'arborescence est **cd**. Si l'on est au niveau de la racine, pour aller à notre répertoire **/home/olivier/linux** on doit taper :

```
cd /home/olivier/linux
```

On a tapé un chemin absolu, on se trouve maintenant sous **/home/olivier/linux**, si l'on veut aller sous **/home/olivier**, on doit taper :

```
cd ..
```

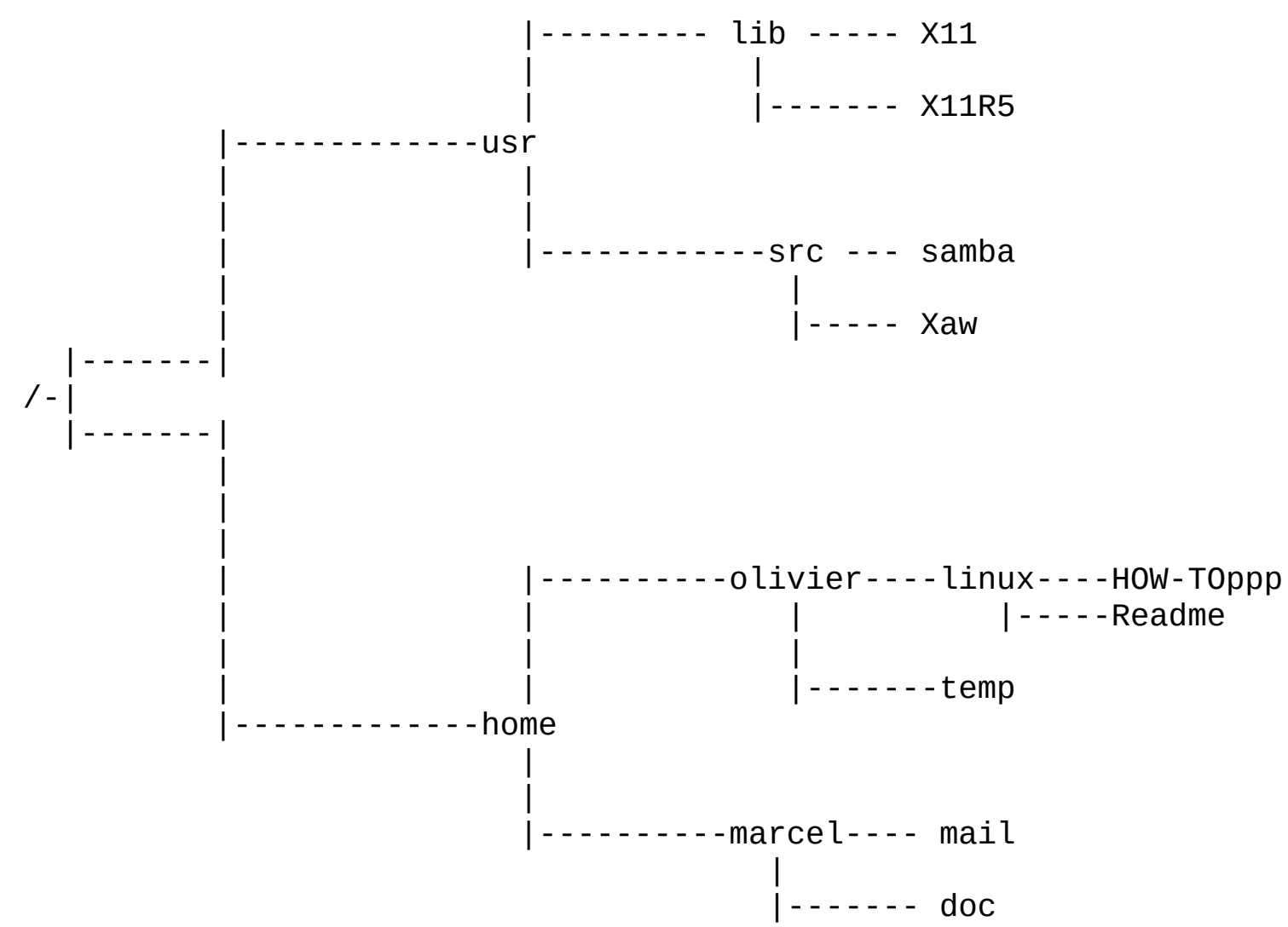
En effet **..** représente le répertoire père, **/home/olivier** étant le répertoire père de **/home/oliver/linux**, en tapant cette commande, on se retrouve à l'endroit désiré.

Si vous voulez connaître ou vous vous trouvez, vous disposez de la commande **pwd**, ainsi si vous tapez **pwd** juste après la séquence de commandes précédentes, vous obtenez :

```
/home/olivier
```

7.3 Visualiser les fichiers

La commande ls permet de visualiser le contenu de répertoires, vous voyez les noms des fichiers présents sous le répertoire.



Si vous allez sous **/home/olivier/linux** (commande **cd /home/olivier/linux**), si vous voulez visualiser les fichiers contenus, vous tapez **ls**, vous obtenez :

```
HOW-TOppp Readme
```

La commande **ls** peut être utilisée avec des options, si précédemment vous aviez tapez **ls -l**, vous auriez obtenu:

```
-rw-rw-rw- 1 olivier users 17356 Dec 3 12:11 HOW-TOppp
-rw-r--r-- 1 olivier users 7432 Nov 21 02:21 Readme
```

La signification des champs est la suivante:

- **-rw-rw-rw-** type de fichier et ses caractéristiques de protection

- **1** le nombre de lien
- **olivier** le nom du propriétaire
- **users** le groupe d'utilisateurs auquel appartient le propriétaire
- **17356** la taille du fichier (en octets)
- **Dec 3** le jour de dernière modification
- **12 :11** l'heure de dernière modification
- **HOW-TOppp** le nom du fichier

Pour avoir ces informations uniquement d'un fichier, vous taperez :

```
ls -l nom-du-fichier
```

A noter que sur beaucoup de système la commande **ll** est équivalent à **ls -l.**

7.4 Commandes de gestion des répertoires

Pour gérer les répertoires, vous disposez des commandes suivantes :

mkdir nom-de-répertoire Création d'un répertoire

rmdir nom-de-répertoire Suppression d'un répertoire vide

mv répertoire répertoire-d'accueil déplacement d'un répertoire

mv répertoire nouveau-nom Changement de nom d'un répertoire

7.5 Commandes de gestion des fichiers

Pour gérer les fichiers vous disposez des commandes suivantes:

touch mon-fichier création d'un fichier vide,

more mon-fichier visualisation d'un fichier page à page,

rm mon-fichier suppression d'un fichier,

mv mon-fichier répertoire d'accueil déplacement d'un fichier,

mv mon-fichier nouveau-nom changement de nom d'un fichier,

cp nom-fichier répertoire-d'accueil/autre-nom copie de fichier,

file mon-fichier pour savoir si on a un fichier binaire (exécutable) ou un fichier texte. On obtient pour un fichier texte, comme sortie mon-fichier : ascii text.

7.6 Les liens

Dans l'arborescence UNIX en tapant la commande **ls -l** on peut rencontrer cette syntaxe un peu particulière.

```
lrwxrwxrwx 1 root root 14 Aug 1 01:58 Mail -> ../../bin/mail*
```

Ca signifie que le fichier **Mail** pointe vers le fichier **mail** qui se trouve dans le répertoire **/bin**, en d'autres termes **Mail** est un lien vers le fichier **mail**.

Un lien est créé pour pouvoir accèder au même fichier à différents endroits de l'arborescence. Sous Windows on retrouve à peu près l'équivalent avec la notion de raccourci.

La commande **ln** (pour link) sert à créer des liens. Par exemple:

```
ln -s /home/olivier/linux/readme /tmp/lisezmoi
```

Le fichier source est **readme** sous **/home/olivier/linux**, le lien créé est lisezmoi sous **/tmp.** En faisant un **man ln**, vous découvrirez qu'il existe des liens hards et softs, sans rentrer dans les détails, je vous conseille dans un premier temps de vous limiter aux liens softs (option **-s**) car les liens hards ne permettents pas de visualiser directement le lien (la petite flèche -> quand on tape **ls -l**).

7.7 Les inodes

Sous un système UNIX, un fichier quel que soit son type est identifié par un numéro appelé numéro d'inode, qu'on pourrait traduire en français par "i-noeud". Ainsi derrière la façade du shell, un répertoire n'est qu'un fichier, identifié aussi par un inode, contenant une liste d'inode représentant chacun un fichier.

La différence entre un lien hard et symbolique se trouve au niveau de l'inode, un lien hard n'a pas d'inode propre, il a l'inode du fichier vers lequel il pointe. Par contre un lien symbolique possède sa propre inode. A noter que vous ne pouvez pas créer de liens hards entre deux partitions de disque différente, vous n'avez pas cette contrainte avec les liens symboliques.

Pour connaître le numéro d'inode d'un fichier, vous pouvez taper:

```
ls -i mon-fichier
```

7.8 Les métacaractères

Si vous êtes à la recherche d'un fichier qui commence par la lettre **a**, en faisant **ls**, vous voudriez voir que les fichiers commençant par **a**. De même si vous voulez appliquer une commande à certains fichiers mais pas à d'autres. C'est le but des métacaractères, ils vous permettent de faire une sélection de fichiers suivant certains critères.

Le métacaractère le plus fréquemment utilisé est *****, il remplace une chaîne de longueur non définie.

Avec le critère *****, vous sélectionnez tous les fichiers. Par le critère **a***, vous sélectionnez tous les fichiers commençant par **a**.

```
ls a*
```

Va lister que les fichiers commençant par **a**. De même ***a** opère une sélection des noms de fichiers se terminant par **a**. Le critère ***a*** va faire une sélection sur les noms de fichiers qui ont le caractère **a** dans leur nom, quelque soit sa place.

Le métacaractère **?** remplace un caractère unique. Avec le critère **a??**, vous sélectionnez les fichiers dont le nom commence par **a**, mais qui contiennent au total trois caractères, exactement.

Les métacaractères **[ ]** représente une série de caractères. Le critère **[aA]*** permet la sélection des fichiers dont le nom commence par un **a** ou **A** (minuscule ou majuscule). Le critère **[a-d]*** fait la sélection des fichiers dont le nom commence par **a** jusqu'à **d**. Le critère ***[de]** fait la sélection des fichiers dont le nom se termine par **d** ou **e**.

Vous voyez donc que les caractères **[]**,***** et **?** sont des caractères spéciaux, qu'on ne peut utiliser comme des simples caractères, parce qu'ils sont interprétés par le shell, comme des métacaractères. Vous pouvez cependant inhiber leur fonctionnement. En tapant :

```
ls mon-fichier?
```

Le shell va interpréter le **?** comme un métacaractère et afficher tous les fichiers qui commencent par **mon-fichier** et qui se termine par un caractère unique quelconque. Si vous ne voulez pas que le **?** soit interprété vous devez taper.

```
ls mon-fichier\  ?
```

---

8. Les droits d'accès

8.1 Identification de l'utilisateur

On a vu auparavant que pour pouvoir se connecter sur une machine, on doit être déclaré sur la machine. Tout utilisateur appartient à un groupe, concrètement dans une université par exemple vous aurez les professeurs dans le groupe enseignant et les élèves dans le groupe élève.

Chaque utilisateur est identifié par un numéro unique UID (User identification), de même chaque groupe est identifié par un numéro unique GID (Group identification).

Vous pouvez voir votre UID et GID en éditant le fichier **/etc/passwd**, c'est respectivement troisième et quatrième champ, après le nom (le login), et le mot de passe crypté.

8.2 Définition des droits d'utilisateur

Cas d'un fichier classique

Avec UNIX les fichiers bénéficient d'une protection en lecture, écriture et exécution, c'est à dire vous pouvez choisir si vous voulez que vos fichiers soient lisibles et/ou modifiables par d'autres, vous pouvez empêcher que d'autres utilisateurs lancent vos exécutables. C'est le principe des droits d'accès.

Nous avons vu qu'en tapant **ls -l** le premier champ correspondait au droit d'accès, on avait une sortie de ce type :

```
-rwxrw-r-- 1 olivier users 34568 Dec 3 14 :34 mon-fichier
```

La signification des lettres **rwx** et la suivante :

r (read) on peut lire le fichier **w** (write) on peut modifier le fichier **x** (exécutable) on peut exécuter le fichier (c'est donc un exécutable) **-** aucun droit autorisé

Le champ **-rwxrw-r--** regroupe les droits du propriétaire du fichier, du groupe auquel appartient le propriétaire et les autres utilisateurs.

- on a affaire à un fichier classique (c'est à ni un répertoire, ni un fichier spécial) **rwx** droits sur le fichier du propriétaire **rw-** droits sur le fichier du groupe auquel appartient le propriétaire (users) **r--** droits sur le fichier des autres utilisateurs (ceux n'appartenant au groupe users)

Par exemple pour notre fichier le propriétaire **olivier** a des droits en écriture, lecture et exécution, le groupe a un droit en lecture et écriture mais aucun droit en exécution, les autres utilisateurs ont uniquement le droit en lecture du fichier.

Pour info le **1** après les droits signifie que le fichier **mon-fichier** n'a aucun lien qui pointe vers lui, si on avait eu **2**, cela signifiait que quelque part dans l'arborescence, il y a un lien qui pointe vers lui, ce nombre s'incrémentant avec le nombre de lien.

Cas d'un répertoire

Pour un répertoire le **x** n'est pas un droit en exécution, mais un droit d'accès au répertoire, sans ce droit, on ne peut pas accéder au répertoire et voir ce qu'il y a dedans.

En tapant **ls -l** sur un répertoire, vous obtenez :

```
drwxr-x--- 1 olivier users 13242 Dec 2 13 :14 mon-répertoire
```

d signifie qu'on a affaire à un répertoire, **rwx** sont les droits du propriétaire **olivier** qui est autorisé en lecture, écriture et droit d'accès au répertoire**r-x** droits du groupe users, autorisé en lecture, droit d'accès au répertoire, pas de droit en écriture **---** droits des autres utilisateurs, aucun droit dans le cas présent

Cas d'un lien

Pour un lien, la signification est similaire à celle d'un fichier classique, à la différence que vous avez un **l** à la place du **-** en tout début de ligne.

```
lrwxrwxrwx 1 root root 14 Aug 1 01:58 Mail -> ../../bin/mail*
```

8.3 Commandes associées

Changer les droits : chmod

La commande **chmod** permet de modifier les droits d'accès d'un fichier (ou répertoire). Pour pouvoir l'utiliser sur un fichier ou un répertoire, il faut en être le propriétaire. La syntaxe est la suivante :

```
chmod utilisateur            opération              droit d'accès
      u propriétaire (user)  +ajout d'un droit      r droit en lecture
      g groupe (group)       -suppression d'un droit w droit en écriture
      o les autres (other)   =ne rien faire         x*
```

* droit en exécution pour un fichier, droit d'accès pour un répertoire.

Exemple vous voulez donner un droit en écriture pour le groupe du fichier **mon-fichier**

```
chmod g+w mon-fichier
```

Pour supprimer le droit d'accès du répertoire **mon-répertoire** aux autres utilisateurs (autres que propriétaire et utilisateurs du groupe)

```
chmod o-x mon-repertoire
```

En tapant

```
chmod u+x,g-w mon-fichier
```

Vous ajoutez le droit en exécution pour le propriétaire, et enlevez le droit en écriture pour le groupe du fichier.

Vous avez une autre méthode pour vous servir de la commande **chmod**. On considère que **r=4**, **w=2** et **x=1**, si vous avez un fichier avec les droits suivants **-rw-rw-rw-**, pour les droits utilisateurs vous avez **(r=)4+(w=)2=6**, de même pour le groupe et les autres. Donc **-rw-rw-rw-** est équivalent à **666**. En suivant la même règle **rwxrw-r-- est** équivalent à 754.

Pour mettre un fichier avec les droits**-r--r--r--** vous pouvez taper :

```
chmod 444 mon-fichier
```

On appelle ce système de notation, la notation octale.

Changer les droits par défaut : umask

Quand vous créer un fichier, par exemple avec la commande **touch**, ce fichier par défaut possède certains droits. Ce sont **666** pour un fichier (**-rw-rw-rw-**) et **777** pour un répertoire (**-rwxrwxrwx**), ce sont les droits maximum. Vous pouvez faire en sorte de changer ces paramètres par défaut. La commande **umask** est là pour ça.

Pour un fichier :

Si vous tapez **umask 022**, vous partez des droits maximum **666** et vous retranchez **022**, on obtient donc **644**, par défaut les fichiers auront comme droit **644** (**-rw-r-r--**).

Si vous tapez **umask 244**, vous partez des droits maximum **666** et vous retranchez **244**, on obtient donc **422**, par défaut les fichiers auront comme droit **422** (**-rw--w--w-**).

Pour un répertoire :

Si vous tapez **umask 022**, vous partez des droits maximum **777** et vous retranchez **022**, on obtient donc **755**, par défaut les fichiers auront comme droit **644** (**-rwxr-xr-x**).

Si vous tapez **umask 244**, vous partez des droits maximum **777** et vous retranchez **244**, on obtient donc **533**, par défaut les fichiers auront comme droit **422** (**-rwx-wx-wx**).

umask n'est utilisatable que si on est propriétaire du fichier.

Changer le propriétaire et le groupe

Vous pouvez " donner " un fichier vous appartenant à un autre utilisateur, c'est à dire qu'il deviendra propriétaire du fichier, et que vous n'aurez plus que les droits que le nouveau propriétaire voudra bien vous donner sur le fichier.

```
chown nouveau-propriétaire nom-fichier
```

Dans le même ordre d'idée vous pouvez changer le groupe.

```
chgrp nouveau-groupe nom-fichier
```

Ces deux commandes ne sont utilisables que si on est propriétaire du fichier.

NOTA : Sur certains UNIX suivant leur configuration, on peut interdire l'usage de ces commandes pour des raisons de sécurité.

9. Gestion des processus

9.1 Les caractéristiques d'un processus

On a vu auparavant, qu'on pouvait à un moment donné avoir plusieurs processus en cours, à un temps donné. Le système doit être capable de les identifier. Pour cela il attribue à chacun d'entre eux, un numéro appelé **PID** (Process Identification).

Un processus peut lui même créer un autre processus, il devient donc un processus parent ou père, et le nouveau processus, un processus enfant. Ce dernier est identifié par son **PID**, et le processus père par son numéro de processus appelé **PPID** (Parent Process Identification).

Tous les processus sont ainsi identifiés par leur **PID**, mais aussi par le **PPID** du processus qui la créé, car tous les processus ont été créés par un autre processus. Oui mais dans tout ça, c'est qui a créé le premier processus ? Le seul qui ne suit pas cette règle est le premier processus lancé sur le système le processus init qui n'a pas de père et qui a pour **PID** 1.

9.2 Visualiser les processus

On peut visualiser les processus qui tournent sur une machine avec la commande : **ps** (options), les options les plus intéressantes sous HP-UX sont **-e** (affichage de tous les processus) et **-f** (affichage détaillée). La commande **ps -ef** donne un truc du genre :

```
UID      PID  PPID  C STIME      TTY      TIME    COMMAND
root     1    0     0 Dec 6      ?        1:02    init
...
jean     319 300    0 10:30:30   ?        0:02    /usr/dt/bin/dtsession
olivier 321 319     0 10:30:34   ttyp1    0:02    csh
olivier 324 321     0 10:32:12   ttyp1    0:00    ps -ef
```

La signification des différentes colonnes est la suivante:

- **UID** nom de l'utilisateur qui a lancé le process
- **PID** correspond au numéro du process
- **PPID** correspond au numéro du process parent
- **C** au facteur de priorité : plus la valeur est grande, plus le processus est prioritaire
- **STIME** correspond à l'heure de lancement du processus
- **TTY** correspond au nom du terminal
- **TIME** correspond à la durée de traitement du processus
- **COMMAND** correspond au nom du processus.

Pour l'exemple donné, à partir d'un shell vous avez lancé la commande **ps -ef**, le premier processus à pour **PID** 321, le deuxième 324. Vous noterez que le **PPID** du process " **ps -ef** " est 321 qui correspond au shell, par conséquent le shell est le process parent, de la commande qu'on vient de taper.

Certains processus sont permanents, c'est à dire qu'ils sont lancés au démarrage du système et arrêtés uniquement à l'arrêt du système. On appelle ces process des daemons, le terme démon est une francisation, daemon sont des abréviations.

Pour voir les process d'un seul utilisateur, vous pouvez taper :

```
ps -u olivier
```

D'un UNIX à l'autre la sortie peut changer. Sous LINUX par exemple **ps -Al** permet une sortie assez riche, en faisant un **man ps**, vous aurez l'éventail de tous les paramètres possibles.

9.3 Commandes de gestion des processus

Changer la priorité d'un processus

Les processus tournent avec un certain degré de priorité, un processus plus prioritaire aura tendance à s'accaparer plus souvent les ressources du système pour arriver le plus vite possible au terme de son exécution. C'est le rôle du système d'exploitation de gérer ces priorités.

Vous disposez de la commande **nice** pour modifier la priorité d'un processus. La syntaxe est la suivante :

```
nice -valeur commande
```

Plus le nombre est grand, plus la priorité est faible. Par exemple une valeur de 0 donne, la priorité la plus haute 20 donne la priorité la plus faible.

La fourchette de valeur dépend de l'UNIX qu'on utilise.

Par exemple :

```
nice -5 ps -ef
```

Généralement on utilise nice sur des commandes qui prennent du temps, sur des commandes courantes l'effet de **nice** est imperceptible. On l'utilisera par exemple pour compiler un programme.

```
nice -5 cc monprogramme.c
```

Arrêter un processus

Vous disposez de la commande **kill** pour arrêter un processus, on doit aussi tuer un processus. Si vous voulez arrêter un processus, vous devez connaître son **PID** (commande ps), puis vous tapez :

```
kill -9 PID
```

Un utilisateur ne peut arrêter que les processus qui lui appartient (qu'il a lancé). Seul l'administrateur système a le droit d'arrêter un processus ne lui appartenant pas.

9.4 Lancer en processus en tâche de fond

Pour lancer une commande quelconque, vous en saisissez le nom après le prompt du shell, tant que la commande n'est pas terminée, vous n'avez plus la main au niveau du shell, vous ne disposez plus du prompt. Si la commande prend un certain temps, votre shell ne vous donnera pas la main tant que la commande n'est pas terminée, vous êtes obligé de lancer un autre shell, pour taper une autre commande.

Vous disposez d'une technique simple qui permet de lancer une commande à partir d'un shell, et de reprendre aussitôt la main. Il vous suffit de rajouter un **&** à la fin de commande. Celle-ci se lancera en " tâche de fond ", et vous reviendrez directement au prompt du shell.

En tapant une commande en tâche de fond, vous aurez à l'affichage :

```
> ps ef &
[321]
>
```

A la suite de la saisie de la commande suivie d'un **&**, le shell vous donne immédiatement la main, et affiche le numéro du **PID** du processus lancé.

En lançant une commande à partir du shell sans le **&** à la fin, et si celle-ci prend du temps à vous rendre la main, vous pouvez faire en sorte qu'elle bascule en tâche de fond, pour que vous repreniez la main.

```
>netscape
```

Vous voulez basculer netscape en tâche de fond tapez, **CTRL+Z**, il va afficher

```
311 stopped +
```

311 étant le **PID** du process netscape. Tapez ensuite **bg** (pour background), vous voyez s'afficher

```
[311]
```

Ca y est votre processus netscape est en tâche de fond et le shell vous rend la main.

---

10. Les titres UNIX

10.1 Modifier les données d'un fichier

Coupe un fichier en morceau : split

La commande **split** permet de couper un fichier en morceau (en plusieurs fichiers), en tapant :

```
split -10 mon-fichier fichier
```

Vous allez créer les fichiers **fichieraa**, **fichierab**, **fichierac**, ... qui contiendront tous 10 lignes. Le premier **fichieraa** contient les 10 premières lignes, ainsi de suite.

Trier des fichiers : sort

Soit le fichier **carnet-adresse** suivant :

```
maurice:29:0298334432:Crozon
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
yvonne:92:0133444335:Palaiseau
```

Le premier champ représente le nom, le deuxième le département, le troisième le numéro de téléphone et le dernier la ville. Attention le premier champ est noté 0, le deuxième 1, ainsi de suite.

En faisant **sort** sans argument :

```
sort carnet-adresse
```

Par défaut il va trier sur le premier caractère et ranger donc dans l'ordre alphabétique :

```
marcel:13:0466342233:Marseille
maurice:29:0298334432:Crozon
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

Si vous voulez trier sur le deuxième champ (le département), vous devez d'abord indiquer que le **:** est le caractère qui sépare deux champs (par défaut c'est l'espace), avec l'option **-t :**. Vous devez ensuite indiquer que vous trier un chiffre avec l'option **-n** (numérique). Pour indiquer qu'on veut trier le deuxième champ, il faut marquer qu'on veut trier à partir du second champ (**+1**) jusqu'au troisième (**-2**). Soit le résultat suivant ;

```
sort -n -t : +1 -2 carnet-adresse
```

On obtient :

```
marcel:13:0466342233:Marseille
maurice:29:0298334432:Crozon
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

Avec la commande :

```
sort -t : +3 -4 +0 carnet-adresse
```

Vous allez trier suivant le quatrième champ (numéro 3), c'est à dire la ville (tri par ordre alphabétique sur le premier caractère), en mettant **+0**, il va effectuer un deuxième tri pour les villes qui commencent par le même caractère, le deuxième tri porte sur le prénom (le premier caractère).

```
maurice:29:0298334432:Crozon
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

Les options de sort sont les suivantes :

- **-b** ignore les espaces et les tabulations en début de champ
- **-d** tri sur les caractères alphanumériques (caractères, chiffres et espace) uniquement
- **-r** inverse l'ordre de tri
- **-f** pas de différence entre minuscule et majuscule
- **-tx** Le caractère **x** est considéré comme séparateur de champ
- **-u** supprime les lignes doublons
- **-n** trie sur des chiffres

En tapant la commande suivante :

```
sort -t : +3.2 +0 carnet-adresse
```

Vous allez effectuer le tri sur le troisième caractère (numéro 2, le premier a pour numéro 0) du quatrième champ (numéro 3). En cas d'égalité du premier tri, on fait un dernier tri sur le premier caractère du prénom. Soit le résultat :

```
yvonne:92:013344433:Palaiseau
maurice:29:0298334432:Crozon
```

```
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
```

Conversion de chaîne de caractère :tr

La commande **tr** permet de convertir une chaîne de caractère en une autre de taille égale. Les options sont les suivantes :

- **-c** Les caractères qui ne sont pas dans la chaîne d'origine sont convertis selon les caractères de la chaîne de destination
- **-d** destruction des caractères appartenant à la chaîne d'origine
- **-s** si la chaîne de destination contient une suite contiguë de caractères identiques, cette suite est réduite à un caractère unique

La commande **tr** a besoin qu'on lui redirige en entrée un fichier, le résultat de la conversion s'affichant sur la sortie standard.

Soit notre fichie**r carnet-adresse** :

```
maurice:29:0298334432:Crozon
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

Pour remplacer le **:** par un **#**, nous taperons :

```
tr " : " " # " < carnet-adresse
```

Pour faire la même chose on peut aussi bien éditer le fichier avec **cat** et rediriger par pipe vers **tr**, en tapant :

```
cat carnet-adresse | tr " : " " # "
```

On peut utiliser des métacaractères. En tapant :

```
cat carnet-adresse | tr " [a-f] " " [A-F] "
```

Vous allez remplacer les caractères de **a** à **f** de minuscule en majuscule. Soit:

```
mAuriCE:29:0298334432:Crozon
mArCEl:13:0466342233:MArsEillE
robErt:75:0144234452:PAris
yvonnE:92:013344433:PAlAisEAu
```

10.2 Edition de fichiers avec critères

Editer un fichier par la fin : tail

Si vous avez un fichier très long, et que vous voulez visualiser que la fin, vous disposez de la commande **tail** :

La syntaxe est la suivante, si vous tapez :

```
tail +10 mon-fichier
```

Vous obtenez toutes les lignes du fichier de la 10eme jusqu'à la fin.

```
tail -10 mon-fichier
```

Vous obtenez les 10 dernières lignes à partir de la fin.

Vous pouvez indiquer si votre unité est la ligne (par défaut), le bloc ou le caractère avec l'option **-t**

```
tail -10 -c mon-fichier
```

Vous obtenez les 10 derniers caractères du fichier.

Editer un fichier par le début : head

Si vous avez un fichier très long, et que vous voulez visualiser que le début, vous disposez de la commande **head** :

La syntaxe est la suivante, si vous tapez :

```
head +10 mon-fichier
```

Vous obtenez toutes les lignes du fichier de la 10eme jusqu'au début.

```
head -10 mon-fichier
```

Vous obtenez les 10 premières lignes à partir du début.

Vous pouvez indiquer si votre unité est la ligne (par défaut), le bloc ou le caractère avec l'option -t

```
head -10 -c mon-fichier
```

Vous obtenez les 10 premiers caractères du fichier.

Compter les lignes d'un fichier : wc

La commande **wc** permet de compter le nombre de ligne d'un fichier, mais aussi le nombre de mot ou de caractères.

```
wc -l mon-fichier
```

Cette commande va donner le nombre de lignes contenues dans le fichier **mon-fichier**. Pour avoir le nombre de mot l'option est **-w**, l'option **-c** compte le nombre de caractères.

La commande **wc** sans option donne à la fois le nombre de ligne, le nombre de caractères et le nombre de mots.

Si vous voulez connaître le nombre de fichier dans un répertoire, la commande sera donc :

```
ls -l | wc -l
```

Edition de champ d'un fichier : cut

La commande **cut** permet d'extraire certains champs d'un fichier. Les options sont les suivantes :

- **-c** extrait suivant le nombre de caractères
- **-f** extrait suivant le nombre de champs
- **-dx** Le caractère **x** est le séparateur de champ

Avec la commande **cut**, contrairement à sort, le premier champ a comme numéro 1, le deuxième 2 est ainsi de suite.

Nous prendrons toujours notre fichier **carnet-adresse** :

```
maurice:29:0298334432:Crozon
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

La commande :

```
cut -c-10 carnet adresse
```

Va extraire les 10 premiers caractères de chaque ligne, on obtient :

```
maurice:29
marcel:13:
robert:75:
yvonne:92:
```

La commande :

```
cut -c2-5 carnet adresse
```

Va extraire les deuxième au cinquième caractère de chaque ligne.

```
auri
arce
ober
vonn
```

La commande :

```
cut -c25-
```

Va extraire du 25eme caractère jusqu'à la fin de chaque ligne.

La commande :

```
cut -d: -f1,4 carnet adresse
```

Va extraire le premier et quatrième champ, le **:** fixant le séparateur de champ. On obtient :

```
maurice:Crozon
marcel:Marseille
robert:Paris
yvonne:Palaiseau
```

La commande :

```
cut -d : -f3- carnet adresse
```

Va extraire du troisième champ jusqu'au dernier champ, soit :

```
0298334432:Crozon
0466342233:Marseille
0144234452:Paris
0133444335:Palaiseau
```

Fusion de fichier : paste

La commande **paste** permet la fusion de lignes de fichiers. Les options sont les suivantes :

- **-dx** Le caractère **x** définit le séparateur de champ
- **-s** Les lignes sont remplacées par des colonnes

Soit le fichier **carnet-adresse** :

```
maurice:29:0298334432:Crozon
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

Et le fichier travail :

```
ingénieur
pâtissier
facteur
vendeuse
```

En tapant la commande :

```
paste -d : carnet-adresse travail
```

Vous obtenez :

```
maurice:29:0298334432:Crozon:ingénieur
marcel:13:0466342233:Marseille:pâtissier
robert:75:0144234452:Paris:facteur
yvonne:92:013344433:Palaiseau:vendeuse
```

Vous pouvez évidemment rediriger le résultat vers un fichier.

Extraction de lignes communes de deux fichiers : comm

Cette commande permet d'extraire les lignes communes à deux fichiers, soit le fichier **carnet-adresse** :

```
maurice:29:0298334432:Crozon
marcel:13:0466342233:Marseille
robert:75:0144234452:Paris
yvonne:92:013344433:Palaiseau
```

Et **carnet-adresse2**

```
olivier:29:0298333242:Brest
marcel:13:0466342233:Marseille
myriam:30:0434214452:Nimes
yvonne:92:013344433:Palaiseau
```

La commande :

```
comm carnet-adresse carnet-adresse2
```

Nous donnera :

```
marcel:13:0466342233:Marseille
yvonne:92:013344433:Palaiseau
```

10.3 Comparaison de fichiers

Comparer deux fichiers : cmp

La commande **cmp** indique si deux fichiers sont identiques. En tapant :

```
cmp fichier1 fichier2
```

Si les deux sont identiques, la commande ne génère aucune sortie, s'ils sont différents la commande indique la position de la première différence (ligne et caractère), avec une sortie du genre :

```
fichier1 fichier2 differ : char 34, line 2
```

Edition des différences entre deux fichiers : diff

Cette commande permet de rechercher les différences entre deux fichiers. La syntaxe est la suivante **diff fichier1 fichier2**, **diff** fait en sorte de vous donner des indications pour que le **fichier1** soit identique au **fichier2**. Soit le fichier **carnet-adresse** :

```
olivier:29:0298333242:Brest
marcel:13:0466342233:Marseille
myriam:30:0434214452:Nimes
yvonne:92:013344433:Palaiseau
toto:12:0434231122:Rodez
```

et **carnet-adresse2**

```
olivier:29:0298333242:Brest
marcel:13:0466342233:Gardagnes
myriam:30:0434214452:Nimes
yvonne:92:013344433:Palaiseau
```

La commande :

```
diff carnet-adresse carnet-adresse2
```

Génère comme sortie :

```
2c2
< marcel:13:0466342233 :Marseille
---
<marcel:13:0466342233 :Gardagnes
5d
>toto :12 :0434231122 :Rodez
```

Ce qui nous indique que pour **carnet-adresse** soit identique à **carnet-adresse2**, il faut que la deuxième ligne du premier fichier soit échangée (c pour change) contre la ligne du second. Il faut aussi supprimer (d pour delete) la cinquième ligne du premier fichier.

Dans d'autres exemples, on pourrait avoir aussi une sortie du genre 10,15c 12,17 ce qui signifie que pour que le premier fichier soit identique au second, les lignes 10 à 15 doivent intégralement échangées contre les lignes 12 à17 du second fichier.

---

11. Les commandes grep et find

11.1 Les expressions régulières

On a vu auparavant ce qu'étaient les métacaractères. Les expressions régulières sont aussi des suites de caractères permettant de faire des sélections. Elles fonctionnent avec certaines commandes comme **grep**.

Les différentes expressions régulières sont :

- `début de ligne`
- `. un caractère quelconque`
- `$ fin de ligne`
- `x* zéro ou plus d'occurrences du caractère x`
- `x+ une ou plus occurrences du caractère x`
- `x? une occurrence unique du caractère x`
- `[...] plage de caractères permis`
- `[...] plage de caractères interdits`
- `\ {n\} pour définir le nombre de répétition n du caractère placé devant`

Exemple l'expression **[a-z][a-z] *** cherche les lignes contenant au minimum un caractère en minuscule. **[a-z]** caractère permis, **[a-z]*** recherche d'occurrence des lettres permises.

L'expression **[0-9]\ {4\}$** a pour signification, du début à la fin du fichier **$**, recherche les nombres**[0-9]** de 4 chiffres \ **{4\}**.

11.2 La commande grep

La commande **grep** permet de rechercher une chaîne de caractères dans un fichier. Les options sont les suivantes :

- `-v affiche les lignes ne contenant pas la chaîne`
- `-c compte le nombre de lignes contenant la chaîne`
- `-n chaque ligne contenant la chaîne est numérotée`
- `-x ligne correspondant exactement à la chaîne`
- `-l affiche le nom des fichiers qui contiennent la chaîne`

Exemple avec le fichier **carnet-adresse** :

```
olivier:29:0298333242:Brest
marcel:13:0466342233:Gardagnes
myriam:30:0434214452:Nimes
yvonne:92:013344433:Palaiseau
```

On peut utiliser les expressions régulières avec **grep**. Si on tape la commande :

```
grep ^[a-d] carnet-adresse
```

On va obtenir tous les lignes commençant par les caractères compris entre a et d. Dans notre exemple, on n'en a pas, d'où l'absence de sortie.

```
grep Brest carnet-adresse
```

Permet d'obtenir les lignes contenant la chaîne de caractère Brest, soit :

```
olivier:29:0298333242:Brest
```

Il existe aussi les commandes fgrep et egrep équivalentes.

11.3 La commande find

Présentation

La commande **find** permet de retrouver des fichiers à partir de certains critères. La syntaxe est la suivante :

```
find <répertoire de recherche> <critères de recherche>
```

Les critères de recherche sont les suivants :

- **-name** recherche sur le nom du fichier,
- **-perm** recherche sur les droits d'accès du fichier,
- **-links** recherche sur le nombre de liens du fichier,
- **-user** recherche sur le propriétaire du fichier,
- **-group** recherche sur le groupe auquel appartient le fichier,
- **-type** recherche sur le type (d=répertoire, c=caractère, f=fichier normal),
- **-size** recherche sur la taille du fichier en nombre de blocs (1 bloc=512octets),
- **-atime** recherche par date de dernier accès en lecture du fichier,
- **-mtime** recherche par date de dernière modification du fichier,
- **-ctime** recherche par date de création du fichier.

On peut combiner les critères avec des opérateurs logiques :

- **critère1 critère2** ou **critère1 -a critère2** correspond au **et** logique,
- **!critère** non logique,
- **\ (critère1 -o critère2\)** ou logique,

La commande **find** doit être utilisé avec l'option **-print**. Sans l'utilisation de cette option, même en cas de réussite dans la recherche, **find** n'affiche rien à la sortie standard (l'écran, plus précisément le shell).

La commande **find** est récursive, c'est à dire où que vous tapiez, il va aller scruter dans les répertoires, et les sous répertoires qu'il contient, et ainsi de suite.

Recherche par nom de fichier

Pour chercher un fichier dont le nom contient la chaîne de caractères **toto** à partir du répertoire **/usr**, vous devez tapez :

```
find /usr -name toto -print
```

En cas de réussite, si le(s) fichier(s) existe(nt), vous aurez comme sortie :

```
toto
```

En cas d'échec, vous n'avez rien.

Pour rechercher tous les fichiers se terminant par **.c** dans le répertoire **/usr**, vous taperez :

```
find /usr -name " *.c " -print
```

Vous obtenez toute la liste des fichiers se terminant par **.c** sous les répertoires contenus dans **/usr** (et dans **/usr** lui même).

Recherche suivant la date de dernière modification

Pour connaître les derniers fichiers modifiés dans les 3 derniers jours dans toute l'arborescence (/), vous devez taper :

```
find / -mtime 3 -print
```

Recherche suivant la taille

Pour connaître dans toute l'arborescence, les fichiers dont la taille dépasse 1Mo (2000 blocs de 512Ko), vous devez taper :

```
find / -size 2000 -print
```

Recherche combinée

Vous pouvez chercher dans toute l'arborescence, les fichiers ordinaires appartenant à olivier, dont la permission est fixée à 755, on obtient :

```
find / -type f -user olivier -perm 755 -print
```

Redirection des messages d'erreur

Vous vous rendrez compte assez rapidement qu'en tant que simple utilisateur, vous n'avez pas forcément le droit d'accès à un certain nombre de répertoires, par conséquent, la commande find peut générer beaucoup de messages d'erreur (du genre permission denied), qui pourraient noyer l'information utile. Pour éviter ceci, vous pouvez rediriger les messages d'erreur dans un fichier poubelle (comme **/dev/null**), les messages d'erreur sont alors perdus (rien ne vous empêche de les sauvegarder dans un fichier, mais ça n'a aucune utilité avec la commande **find**).

```
find . -name bobo -print
```

Recherche en utilisant les opérateurs logiques

Si vous voulez connaître les fichiers n'appartenant pas à l'utilisateur **olivier**, vous taperez :

```
find . ! -user olivier -print
```

! -user olivier, est la négation de **-user olivier**, c'est à dire c'est tous les utilisateurs sauf **olivier**.

Recherche des fichiers qui ont pour nom **a.out** et des fichiers se terminant par **.c**. On tape :

```
find . \  ( -name a.out -o -name " *.c " \  ) -print
```

On recherche donc les fichiers dont le nom est **a.out** ou les fichiers se terminant par ***.c**, une condition ou l'autre.

Recherche des fichiers qui obéissent à la fois à la condition a pour nom **core** et à la condition a une taille supérieure à 1Mo.

```
find . \  (-name core -a size +2000 \  ) -print
```

Les commandes en option

L'option **-print** est une commande que l'on passe à **find** pour afficher les résultats à la sortie standard. En dehors de **print**, on dispose de l'option **-exec**. **find** couplé avec **exec** permet d'exécuter une commande sur les fichiers trouvés d'après les critères de recherche fixés. Cette option attend comme argument une commande, celle ci doit être suivi de **{}\ ;**.

Exemple recherche des fichiers ayant pour nom **core**, suivi de l'effacement de ces fichiers.

```
find . -name core -exec rm {}\   ;
```

Tous les fichiers ayant pour nom core seront détruits, pour avoir une demande de confirmation avant l'exécution de rm, vous pouvez taper :

```
find . -name core -ok rm {}\   ;
```

Autres subtilités

Une fonction intéressante de **find** est de pouvoir être utilisé avec d'autres commandes UNIX. Par exemple:

```
find . -type f -print | xargs grep toto
```

En tapant cette commande vous allez rechercher dans le répertoire courant tous les fichiers normaux (sans les répertoires, fichiers spéciaux), et rechercher dans ces fichiers tous ceux contenant la chaîne **toto**.

Next Previous Contents

12. Expressions régulières et sed

12.1 Les expressions régulières

Présentation

Une expression régulière (en anglais Regular Expression ou RE) sert à identifier une chaîne de caractère répondant à un certain critère (par exemple chaîne contenant des lettres minuscules uniquement). L'avantage d'une expression régulière est qu'avec une seule commande on peut réaliser un grand nombre de tâche qui seraient fastidieuses à faire avec des commandes UNIX classiques.

Les commandes ed, vi, ex, sed, **awk**, **expr** et **grep** utilisent les expressions régulières.

L'exemple le plus simple d'une expression régulière est une chaîne de caractères quelconque **toto** par exemple. Cette simple expression régulière va identifier la prochaine ligne du fichier à traiter contenant une chaîne de caractère correspondant à l'expression régulière.

Si l'on veut chercher une chaîne de caractère au sein de laquelle se trouve un caractère spécial (/, *****, **$**, **.**, **[**, **]**, **{**, **}**, **!**, entre autres) (appelé aussi métacaractère), on peut faire en sorte que ce caractère ne soit pas interprété comme un caractère spécial mais comme un simple caractère. Pour cela vous devez le faire précéder par \ (backslash). Ainsi si votre chaîne est **/dev**, pour que le **/** ne soit pas interprété comme un caractère spécial, vous devez tapez \ /dev pour l'expression régulière.

Le métacaractère .

Le métacaractère **.** remplace dans une expression régulière un caractère unique, à l'exception du caractère retour chariot (**\ n**). Par exemple chaine. va identifier toutes les lignes contenant la chaine chaine suivit d'un caractère quelconque unique. Si vous voulez identifier les lignes contenant la chaîne **.cshrc**, l'expression régulière correspondante est **\ .cshrc**

Les métacaractères [ ]

Les métacaractères **[]** permettent de désigner des caractères compris dans un certain intervalle de valeur à une position déterminée d'une chaîne de caractères. Par exemple **[Ff]raise** va identifier les chaînes **Fraise** ou **fraise**, **[a-z]toto** va identifier une chaîne de caractère commençant par une lettre minuscule (intervalle de valeur de **a** à **z**) et suivi de la chaîne **toto** (**atoto**, **btoto**, ..., **ztoto**).

D'une manière plus générale voici comment **[]** peuvent être utilisés.

[A-D] intervalle de **A** à **D** (**A**, **B**, **C**, **D**) **par exemple bof[A-D] donne bofA**, **bofB**, **bofC**, **bofD**

[2-5] intervalle de **2** à **5** (**2**, **3**, **4**, **5**) par exemple **12[2-5]2 donne 1222**, **1232**, **1242**, **1252**

[2-56] intervalle de **2** à **5** et **6** (et non pas **56**) (**2**, **3**, **4**, **5**, **6**) **par exemple 12[2-56]2 donne 1222**, **1232**, **1242**, **1252**, **1262**

[a-dA-D] intervalle de **a** à **d** et **A** à **D** (**a**, **b**, **c**, **d**, **A**, **B**, **C**, **D**) **par exemple z[a-dA-D]y donne zay**, **zby**, **zcy**, **zdy**, **zAy**, **zBy**, **zCy**, **zDy**

[1-3-] intervalle de **1** à **3** et **-** (**1**, **2**, **3**, **-**) **par exemple [1-3-]3 donne 13**, **23**, **33**, **-3**

[a-cI-K1-3] intervalle de **a** à **c**, **I** à **K** et **1** à **3** (**a**, **b**, **c**, **I**, **J**, **K**, **1**, **2**, **3**)

On peut utiliser **[]** avec un pour identifier le complèment de l'expression régulière. En français pour identifier l'opposé de l'expression régulière. Vous avez toujours pas compris ? Voici un exemple: **[0-9]toto** identifie les lignes contenant une chaîne **toto**, le caractère juste avant ne doit pas être un chiffre (exemple **atoto**, **gtoto** mais pas **1toto**, **5toto**). Autre exemple **[a-zA-Z]** n'importe quel caractère sauf une lettre minuscule ou majuscule. Attention à la place de , si vous tapez **[1-3]**, c'est équivalent aux caractères **1**, **2**, **3** et .

Les métacaractères et $

Le métacaractère identifie un début de ligne. Par exemple l'expression régulière **a** va identifier les lignes commençant par le caractère **a**.

Le métacaractère **$** identifie une fin de ligne. Par exemple l'expression régulière **a$** va identifier les lignes se terminant par le caractère **a**.

L'expression régulière **chaine$** identifie les lignes qui contiennent strictement la chaîne **chaine**.

L'expression régulière **$** identifie une ligne vide.

Le métacaractère *

Le métacaractère ***** est le caractère de répétition.

L'expression régulière **a*** correspond aux lignes comportant 0 ou plusieurs caractère **a**. Son utilisation est à proscrire, car toutes les lignes, même celles ne contenant pas le caractère **a**, répondent aux critères de recherche. **x*** est une source de problèmes, il vaut mieux éviter de l'employer.

L'expression régulière **aa*** correspond aux lignes comportant 1 ou plusieurs caractères **a**.

L'expression régulière **.*** correspond à n'importe quelle chaîne de caractères.

L'expression régulière **[a-z][a-z]*** va chercher les chaînes de caractères contenant 1 ou plusieurs lettres minuscules (de **a** à **z**).

L'expression régulière **[ ][ ]*** est équivalent à tout sauf un blanc.

Les métacaractères \ (\)

Pour le traitement complexe de fichier, il est utile parfois d'identifier un certain type de chaîne pour pouvoir s'en servir dans la suite du traitement comme un sous programme. C'est le principe des sous chaînes, pour mémoriser une sous chaîne, on utilise la syntaxe \ **(expression régulière)**\, cette sous chaîne sera identifié par un chiffre compris par 1 et 9 (suivant l'ordre de définition).

Par exemple \ **([a-z][a-z]*)**\ est une sous chaîne identifiant les lignes contenant une ou plusieurs lettres minuscules, pour faire appel à cette sous chaîne, on pourra utiliser \ **1**. Voir dans le paragraphe **sed** pour un exemple.

12.2 La commande sed

Présentation

sed est éditeur ligne non interactif, il lit les lignes d'un fichier une à une (ou provenant de l'entrée standard) leur applique un certain nombre de commandes d'édition et renvoie les lignes résultantes sur la sortie standard. Il ne modifie pas le fichier traité, il écrit tout sur la sortie standard.

sed est une évolution de l'éditeur **ed** lui même précurseur de **vi**, la syntaxe n'est franchement pas très conviviale, mais il permet de réaliser des commandes complexes sur des gros fichiers.

La syntaxe de **sed** est la suivante:

```
sed -e 'programme sed' fichier-a-traiter
```

ou

```
sed -f fichier-programme fichier-a-traiter
```

Vous disposez de l'option **-n** qui supprime la sortie standard par défaut, **sed** va écrire uniquement les lignes concernées par le traitement (sinon il écrit tout même les lignes non traitées). L'option **-e** n'est pas nécessaire quand vous avez une seule fonction d'édition.

La commande **sed** est une commande très riche, ne vous sont présentées ici que les fonctions les plus courantes, pour plus de détails faites un **man sed** et/ou **man ed**.

La fonction de substitution s

La fonction de substitution **s** permet de changer la première ou toutes les occurences d'une chaîne par une autre. La syntaxe est la suivante:

sed "s/toto/TOTO/" fichier va changer la première occurence de la chaîne **toto** par **TOTO** (la première chaîne **toto** rencontrée dans le texte uniquement)

sed "s/toto/TOTO/3" fichier va changer la troisième occurence de la chaîne **toto** par **TOTO** (la troisième chaîne **toto** rencontrée dans le texte uniquement)

sed "s/toto/TOTO/g" fichier va changer toutes les occurences de la chaîne **toto** par **TOTO** (toutes les chaînes **toto** rencontrées sont changées

sed "s/toto/TOTO/p" fichier en cas de remplacement la ligne concernée est affichée sur la sortie standard (uniquement en cas de substitution)

sed "s/toto/TOTO/w resultat" fichier en cas de substitution la ligne en entrée est inscrite dans un fichier résultat

La fonction de substitution peut évidemment être utilisée avec une expression régulière.

sed -e "s/[Ff]raise/FRAISE/g" fichier substitue toutes les chaînes **Fraise** ou **fraise** par **FRAISE**

La fonction de suppression d

La fonction de suppression **d** supprime les lignes comprises dans un intervalle donné. La syntaxe est la suivante:

```
sed "20,30d" fichier
```

Cette commande va supprimer les lignes 20 à 30 du fichier fichier. On peut utiliser les expressions régulières:

```
sed "/toto/d" fichier
```

Cette commande supprime les lignes contenant la chaîne **toto**. Si au contraire on ne veut pas effacer les lignes contenant la chaîne **toto** (toutes les autres sont supprimées), on tapera:

```
sed "/toto/!d" fichier
```

En fait les lignes du fichier d'entrée ne sont pas supprimées, elles le sont au niveau de la sortie standard.

Les fonctions p, l, et =

La commande **p** (print) affiche la ligne sélectionnée sur la sortie standard. Elle invalide l'option **-n**.

La commande **l** (list) affiche la ligne sélectionnée sur la sortie standard avec en plus les caractères de contrôles en clair avec leur code ASCII (deux chiffres en octal).

La commande = donne le numéro de la ligne sélectionnée sur la sortie standard.

Ces trois commandes sont utiles pour le débogage, quand vous mettez au point vos programmes **sed**.

```
sed "/toto/=" fichier
```

Cette commande va afficher le numéro de la ligne contenant la chaîne **toto**.

Les fonctions q, r et w

La fonction **q** (quit) va interrompre l'exécution de **sed**, la ligne en cours de traitement est affichée sur la sortie standard (uniquement si **-n** n'a pas été utilisée).

La fonction **r** (read) lit le contenu d'un fichier et écrit le contenu sur la sortie standard.

La fonction **w** (write) écrit la ligne sélectionnée dans un fichier.

```
sed "/^toto/w resultat" fichier
```

Cette commande va écrire dans le fichier resultat toutes les lignes du fichier **fichier** commençant par la chaîne **toto**.

Les fonctions a et i

La fonction **a** (append) va placer un texte après la ligne sélectionnée. La syntaxe est la suivante:

```
a\
le texte
```

La fonction **i** (insert) va placer un texte avant la ligne sélectionnée. La syntaxe est la suivante:

```
i\
le texte
```

Si votre texte tient sur plusieurs lignes la syntaxe pour le texte est la suivante:

```
ligne 1 du texte\
ligne 2 du texte  \
ligne n du texte  \
dernière ligne
```

Concrètement vous pouvez appeler la fonction **i** ou **a** dans un fichier de commande de **sed**. Par exemple, soit votre fichier **prog.sed** suivant:

```
1i\
début du traitement
s/[tT]oto/TOTO/g
$a  \     fin du traitement\
de notre fichier
```

On exécute la commande en tapant:

```
sed -f prog.sed fichier-a-traiter
```

prog.sed a pour effet d'inscrire avant la première ligne (**1i**) le texte "**début de traitement**", et après la dernière ligne (**$a**) le texte "**fin du traitement** (retour à la ligne) **de notre fichier**".

sed et les sous chaînes

La commande:

```
sed -e "s/\  ([0-9][0-9]*\  )/aa\  1aa/" fichier
```

La sous expression (sous chaîne) **\ ([0-9][0-9]*\)** désigne un ou plusieurs chiffres, chacun sera entouré des caractères **aa**. La chaîne **to2to** deviendra **toaa2aato**.

Next Previous Contents

13. La commande awk

13.1 Présentation

Présentation et syntaxe

awk est une commande très puissante, c'est un langage de programmation a elle tout seule qui permet une recherche de chaînes et l'exécution d'actions sur les lignes sélectionnées. Elle est utile pour récuperer de l'information, générer des rapports, transformer des données entre autres.

Une grande partie de la syntaxe a été empruntée au langage c, d'ailleurs **awk** sont les abréviations de ces 3 créateurs dont k pour Kernighan, un des inventeurs du c.

La syntaxe de **awk** est la suivante:

```
awk [-F] [-v var=valeur] 'programme' fichier
```

ou

```
awk [-F] [-v var=valeur] -f fichier-config fichier
```

L'argument **-F** doit être suivi du séparateur de champ (**-F:** pour un "**:**" comme séparateur de champ).

L'argument **-f** suivi du nom du fichier de configuration de awk.

L'argument **-v** définit une variable (**var** dans l'exemple) qui sera utilisée par la suite dans le programme.

Un programme **awk** possède la structure suivante: **critère de sélection d'une chaîne {action}**, quand il n'y a pas de critère c'est que l'action s'applique à toutes les lignes du fichier.

Exemple:

```
awk -F":" '{print $NF}' /etc/passwd
```

Il n'y a pas de critères, donc l'action s'applique à toutes les lignes du fichier **/etc/passwd**. L'action consiste à afficher le nombre de champ du fichier. **NF** est une variable prédéfinie d'awk, elle est égale au nombre de champs dans une ligne.

Généralement on utilisera **awk** en utilisant un script.

```
#!/bin/sh
awk [-F] [-v var=valeur] 'programme' $1
```

Vous appelerez votre script **mon-script.awk**, lui donnerez des droits en exécution (**755** par exemple), et l'appelerez ainsi:

```
mon-script.awk fichier-a-traiter
```

Dans la suite du cours, on utilisera **awk** en sous entendant que celui-ci est à insérer dans un script.

Le quote ' se trouve sur un clavier azerty standard avec le 4 et éventuellement l'accolade gauche.

ATTENTION: ils existent plusieurs "variétés" de **awk**, il se pourrait que certaines fonctions ou variables systèmes qui vous sont présentées dans ce cours n'existent pas sur votre UNIX. Faites en sorte si vos scripts awk doivent fonctionner sur des plates-formes différentes d'utiliser **gawk** sous licence GNU qui est totalement POSIX.

J'ai constaté des grosses différences de comportement entre le awk natif qu'on soit sous HP-UX, Solaris et sous LINUX, de même quand on insère la commande dans un script, on fait appel à un shell, suivant son type (bash shell, csh, ksh, ...), vous

pouvez avoir quelques surprises.

Enregistrements et champs

awk scinde les données d'entrée en enregistrements et les enregistrements en champ. Un enregistrement est une chaîne d'entrée délimitée par un retour chariot, un champ est une chaîne délimitée par un espace dans un enregistrement.

Par exemple si le fichier à traiter est **/etc/passwd**, le caractère de séparation étant "**:**", un enregistrement est une ligne du fichier, et un champ correspond au chaîne de caractère séparé par un "**:**" (login:mot de passe crypté:UID:GID:commentaires:home directory:shell).

Dans un enregistrement les champs sont référencés par **$1**, **$2**, ..., **$NF** (dernier champ). Par exemple pour **/etc/passwd $1** correspond au login, **$2** au mot de passe crypté, **$3** à l'UID, et **$NF** (ou **$7**) au shell.

L'enregistrement complet (une ligne d'un fichier) est référencé par **$0**.

Par exemple, si l'on veut voir les champs login et home directory de **/etc/passwd**, on tapera:

```
awk -F":" '{print $1,$6}' /etc/passwd
```

13.2 Critères de sélection

Présentation

Un critère peut être une expression régulière, une expression ayant une valeur chaîne de caractères, une expression arithmétique, une combinaison des expressions précédentes.

Le critère est inséré entre les chaînes **BEGIN** et **END**, avec la syntaxe suivante:

```
awk -F":" 'BEGIN{instructions} critères END{instructions}' fichier
```

BEGIN peut être suivi d'instruction comme une ligne de commentaire ou pour définir le séparateur. Exemple **BEGIN { print"Vérification d'un fichier"; FS=":"}**. Le texte à afficher peut être un résumé de l'action de **awk**. De même pour **END** on peut avoir **END{print "travail terminé"}** qui indiquera que la commande a achevé son travail. Le **END** n'est pas obligatoire, de même que le **BEGIN**.

Les expressions régulières

La syntaxe est la suivante:

```
/expression régulière/ {instructions}
$0  /expression régulière/ {instructions}
```

les instructions sont exécutées pour chaque ligne contenant une chaîne satisfaisant à l'expression régulière.

```
expression  /expression régulière/{instructions}
```

les instructions sont exécutées pour chaque ligne où la valeur chaîne de l'expression contient une chaîne satisfaisant à l'expression régulière.

```
expression !/expression régulière/ {instructions}
```

les instructions sont exécutées pour chaque ligne où la valeur chaîne de l'expression ne contient pas une chaîne satisfaisant à l'expression régulière.

Soit le fichier adresse suivant (nom, numéro de téléphone domicile, numéro de portable, numéro quelconque):

```
gwenael | 0298452223 | 0638431234 | 50
marcel | 0466442312 | 0638453211 | 31
judith | 0154674487 | 0645227937 | 23
```

L'exemple suivant vérifie que dans le fichier le numéro de téléphone domicile (champ 2) et le numéro de portable (champ 3) sont bien des nombres.

```
awk 'BEGIN { print "On vérifie les numéros de téléphone; FS="|"}
   $2 ! /^[0-9][0-9]*$/ { print "Erreur sur le numéro de téléphone domicile, ligne
n°"NR": \  n"$0}
   $3 ! /^[0-9][0-9]*$/ { print "Erreur sur le numéro de téléphone du portable, ligne
n°"NR": \  n"$0}
END { print "Vérification terminé"} ' adresse
```

BEGIN est suivi d'une instruction d'affichage qui résume la fonction de la commande, et de la définition du séparateur de champ. L'expression **$2** se référe au deuxième champ d'une ligne (enregistrement) de adresse soit le numéro de téléphone domicile, on recherche ceux qui ne contiennent pas de chiffre (négation de contient des chiffres), en cas de succés on affichera un message d'erreur, le numéro de ligne courante, un retour à la ligne, puis le contenu entier de la ligne. L'expression **$3** se référe au troisième champ d'une ligne (enregistrement) de adresse soit le numéro du portable, on recherche ceux qui ne contiennent pas de chiffre (négation de contient des chiffres), en cas de succés on affichera un message d'erreur, le numéro de ligne courante, un retour à la ligne, puis le contenu entier de la ligne. **END** est suivi d'une instruction d'affichage indiquant la fin du travail.

Les expressions relationnelles

Un critère peut contenir des opérateurs de comparaison (- <, <=,==,**!**=,>=,>). Exemple avec le fichier adresse suivant:

```
awk 'BEGIN { print "On cherche lignes dont le numéro (champ 4) est supérieur à 30";
FS="|"}
 $4 > 30 { print "Numéro supérieur à 30 à la ligne n°"NR": \  n"$0}
END { print "Vérification terminé"} ' adresse
```

Combinaison de critères

Un critère peut être constitué par une combinaison booléenne avec les opérateurs ou (||), et (**&&**) et non (**!**). Exemple:

```
awk 'BEGIN { print "On cherche la ligne avec judith ou avec un numéro inférieur à
30"; FS="|"}
 $1 = = "judith" || $4 < 30 { print "Personne "$1" numéro "$4" ligne n°"NR": \  n"$0}
END { print "Vérification terminé"} ' adresse
```

Plage d'enregistrement délimitées par des critères

La syntaxe est la suivante **critère1,critère2 {instructions}**. Les instructions sont exécutées pour toute les lignes entre la ligne répondant au critère1 et celle au critère2. L'action est exécutée pour les lignes comprises entre la ligne 2 et 6.

```
awk 'BEGIN NR==2;NR==6 { print "ligne n°"NR":\  n"$0}
END ' adresse
```

13.3 Les actions

Présentation

Les actions permettent de transformer ou de manipuler les données, elles contiennent une ou plusieurs instructions. Les actions peuvent être de différents types: fonctions prédéfinies, fonctions de contrôle, fonctions d'affectation, fonctions d'affichage.

Fonctions prédéfinies traitant des numériques

atan2(y,x) arctangente de x/y en radian (entre -pi et pi)

cos(x) cosinus (radian)

exp(x) exponentielle à la puissance x

int(x) partie entière

log(x) logarithme naturel

rand(x) nombre aléatoire (entre 0 et 1)

sin(x) sinus (radian)

sqr(t) racine carrée

srand(x) définition d'une valeur de départ pour générer un nombre aléatoire

Fonctions prédéfinies traitant de chaînes de caractères

Pour avoir la liste des fonctions prédéfinies sur votre plate-forme vous devez faire un **man awk**, voici la liste des fonctions les plus courantes sur un système UNIX.

gsub(expression-régulière,nouvelle-chaine,chaine-de-caractères) dans chaine-de-caractères tous les caractères décrits par l'expression régulière sont remplacés par nouvelle-chaine. gsub et équivalent à gensub.

gsub(/a/,"ai",oi") Remplace la chaine oi par ai

index(chaine-de-caractères,caractère-à-rechercher) donne la première occurence du caractère-à-rechercher dans la chaine chaine-de-caractères

n=index("patate","ta") n=3 length(chaine-de-caractères) renvoie la longueur de la chaine-de-caractères

n=length("patate") n=6

match(chaine-de-caractères,expression-régulière) renvoie l'indice de la position de la chaîne chaine-de-caractères, repositionne RSTART et RLENGTH

n=match("PO1235D",/[0-9][0-9]/) n=3, RSTART=3 et RLENGTH=4

printf(format,valeur) permet d'envoyer des affichages (sorties) formatées, la syntaxe est identique de la même fonction en C

printf("La variable i est égale à %7,2f",i) sortie du chiffre i avec 7 caractères (éventuellement caractères vides devant) et 2 chiffres après la virgule.

printf("La ligne est %s",$0) > "fichier.int" Redirection de la sortie vers un fichier avec >, on peut utiliser aussi la redirection >>. Veillez à ne pas oublier les "" autour du nom du fichier.

split(chaine-de-caractères,tableau,séparateur) scinde la chaîne chaine-de-caractères dans un tableau, le séparateur de champ est le troisième argument

n=split("zorro est arrivé",tab," ") tab[1]="zorro", tab[2]="est", tab[3]="arrivé", n=3 correspond au nombre d'éléments dans le tableau

sprintf(format,valeur) printf permet d'afficher à l'écran alors que sprintf renvoie la sortie vers une chaîne de caractères.

machaine=sprintf("J'ai %d patates",i) machaine="J'ai 3 patates" (si i=3)

substr(chaine-de-caractères,pos,long) Extrait une chaine de longueur long dans la chaîne chaine-de-caractères à partir de la position pos et l'affecte à une chaîne.

machaine=substr("Zorro est arrivé",5,3) machaine="o e"

sub(expression-régulière,nouvelle-chaine,chaine-de-caractères) idem que gsub sauf que seul la première occurence est remplacée (gsub=globale sub)

system(chaine-de-caractères) permet de lancer des commandes d'autres programmes

commande=sprintf("ls | grep toto") Exécution de la commande UNIX "ls |grep toto"

system(commande)

tolower(chaine-de-caracteres) retourne la chaîne de caractères convertie en minuscule

toupper(chaine-de-caracteres) retourne la chaîne de caractères convetie en majuscule

Fonctions définies par l'utilisateur

Vous pouvez définir une fonction utilisateur de telle sorte qu'elle puisse être considérée comme une fonction prédéfinie. La syntaxe est la suivant:

```
fonction mafonction(liste des paramètres)
{
  instructions
  return valeur
}
```

13.4 Les variables et opérations sur les variables

Présentation

On trouve les variables système et les variables utilisateurs. Les variables systèmes non modifiables donnent des informations sur le déroulement du programme. Les variables utilisateurs sont définies par l'utilisateur.

Les variables utilisateur

Le nom des variables est formé de lettres, de chiffres (sauf le premier caractère de la variable), d'underscore. Ce n'est pas nécessaire d'initialiser une variable, par défaut, si c'est un numérique, elle est égale à 0, si c'est une chaîne, elle est égale à une chaîne vide. Une variable peut contenir du texte, puis un chiffre, en fonction de son utilisation **awk** va déterminer son type (numérique ou chaîne).

Les variables prédéfinies (système)

Les variables prédéfinies sont les suivantes (en italique les valeurs par défaut):

ARGC nombre d'arguments de la ligne de commande *néant*

ARGIND index du tableau **ARGV** du fichier courant

ARGV tableau des arguments de la ligne de commande *néant*

CONVFMT format de conversion pour les nombres *"%.6g"*

ENVIRON tableau contenant les valeurs de l'environnement courant

ERRNO contient une chaîne décrivant une erreur *""*

FIELIWIDTHS variable expérimentale à ne pas utiliser

FILENAME nom du fichier d'entrée *néant*

FNR numéro d'enregistrement dans le fichier courant *néant*

FS contrôle le séparateur des champs d'entrée " "

IGNORECASE contrôle les expressions régulières et les opérations sur les chaînes de caractères *0*

NF nombre de champs dans l'enregistrement courant *néant*

NR nombre d'enregistrements lus jusqu'alors néant

OFMT format de sortie des nombres (nombre après la virgule) *"%.6g"*

OFS séparateur des champs de sortie " "

ORS séparateur des enregistrements de sortie

RLENGTH néant longueur de la chaîne sélectionnée par le critère "\ *n*"

RS contrôle le séparateur des enregistrements d'entrée "\ *n*"

RSTART début de la chaîne sélectionnée par le critère *néant*

SUBSEP séparateur d'indiçage "\ *034*"

Opérations sur les variables

On peut manipuler les variables et leur faire subir certaines opérations. On trouve différents types d'opérateurs, les opérateurs arithmétiques classiques (+, -, *****, /, **%**(modulo, reste de la division), (puissance)), les opérateurs d'affectation (=, +=, -=, *****=, /=, **%**=, =). Exemples:

var=30 affectation du chiffre 30 à var

var="toto" affectation de la chaîne toto à var

var="toto " "est grand" concaténation des chaînes

"toto " et "est grand", résultat dans var

var=var-valeur var-=valeur expressions équivalentes

var=var+valeur var+=valeur expressions équivalentes

var=var*valeur var*=valeur expressions équivalentes

var=var/valeur var/=valeur expressions équivalentes

var=var%valeur var%=valeur expressions équivalentes

var=var+1 var++ expressions équivalentes

var=var-1 var-- expressions équivalentes

Les variables de champ

Comme on l'a déjà vu auparavant les champs d'un enregistrement (ligne) sont désignés par **$1**, **$2**,...**$NF**(dernier champ d'une ligne). L'enregistrement complet (ou ligne) est désigné par **$0**. Une variable champ est a les mêmes propriétés que les autres variables. Le signe **$** peut être suivi par une variable, exemple:

```
i=3
print $(i+1)
```

Provoque l'affichage du champ 4

Lorsque **$0** est modifié, automatiquement les variables de champs **$1**,..**$NF** sont redéfinies.

Quand l'une des variables de champ est modifée, **$0** est modifié. ATTENTION le séparateur ne sera pas celui définit par **FS** mais celui définit par **OFS** (output field separator). Exemple:

```
awk 'BEGIN{print" # Tous les utilisateurs du groupe users(GID 22)basculeront dans le
```

```
groupe boulot(GID 24)
          ";FS=":";OFS=":"}
    $4 != 22 {print $0} # Si le groupe n'est pas users on fait rien
    $4 ==22 {$4=24;print $0} # Si le groupe est 22, on lui réaffecte 24
END {print"C'est fini"}' /etc/passwd > passwd.essai
```

13.5 Les structures de contrôle

Présentation

Parmi les structures de contrôle, on distingue les décisions (**if**, **else**), les boucles (**while**, **for**, **do-while**, **loop**) et les sauts (**next**, **exit**, **continue**, **break**).

Les décisions (if, else)

La syntaxe est la suivante:

```
if  (condition) si la condition est satisfaite (vraie)
  instruction1 on exécute l'instruction
else sinon
  instruction2 on exécute l'instruction 2
```

Si vous avez une suite d'instructions à exécuter, vous devez les regrouper entre deux accolades. Exemple:

```
awk ' BEGIN{print"test de l'absence de mot de passe";FS=":"}
NF==7
{ #pour toutes les lignes contenant 7 champss
  if ($2=="") # si le deuxième champ est vide (correspond au mot de passe crypté))
  { print $1 " n'a pas de mot de passe"} # on affiche le nom de l'utilisateur
($1=login) qui n'a pas de mot de passe
  else sinon
  { print $1 " a un mot de passe"} # on affiche le nom de l'utilisateur possédant un
mot de passe
}
END{print"C'est fini") ' /etc/passwd
```

Les boucles (while, for, do-while)

while, **for** et **do-while** sont issus du langage de programmation C. La syntaxe de **while** est la suivante:

```
while  (condition) tant que la condition est satisfaite (vraie)
instruction on exécute l'instruction
```

Exemple:

```
awk ' BEGIN{print"affichage de tous les champs de passwd";FS=":"}
{ i=11 # initialisation du compteur à 1 (on commence par le champ 1)
while(i=<NF) # tant qu'on n'est pas en fin de ligne
  { print $ii # on affiche le champ
  i+++ # incrémentation du compteur pour passer d'un champ au suivant
  }
}
END{print"C'est fini") ' /etc/passwd
```

La syntaxe de **for** est la suivante:

for (instruction de départ; condition; instruction d'incrémentation) On part d'une instruction de départ, on incrémente instruction on exécute l'instruction jusqu'à que la condition soit satisfaite

Exemple:

```
awk ' BEGIN{print" affichage de tous les champs de passwd";FS=":"}
{
for (i=1;i=><NF;i++) # initialisation du compteur à 1, on incrémente le compteur
jusqu'à ce qu'on atteigne NF (fin de la ligne)
  { print $i } # on affiche le champ tant que la condition n'est pas satisfaite
}
END{print"C'est fini") ' /etc/passwd
```

Avec **for** on peut travailler avec des tableaux. Soit le tableau suivant: **tab[1]="patate"**, **tab[2]="courgette"**,**tab[3]="poivron"**. Pour afficher le contenu de chacun des éléments du tableau on écrira:

```
for (index in tab)
{
 print "legume :" tab[index]
}
```

La syntaxe de **do-while** est la suivante:

```
do
{instructions}} on exécute les instructions
while  (condition) jusqu'à que la condition soit satisfaite
```

La différence avec while, est qu'on est sûr que l'instruction est exécutée au moins une fois.

Sauts controlés (break, continue, next, exit)

break permet de sortir d'une boucle, la syntaxe est la suivante:

```
for  (;;;) boucle infinie
{instructions on exécute les instructionss
if  (condition)  break si la condition est satisfaite on sort de la boucle
instructions}
```

Alors que **break** permet de sortir d'une boucle, **continue** force un nouveau passage dans une boucle.

next permet d'interrompre le traitrement sur la ligne courante et de passer à la ligne suivante (enregistrement suivant).

exit permet d'abandonner la commande **awk**, les instructions suivant **END** sont exécutées (s'il y en a).

13.6 Les tableaux

Présentation

Un tableau est une variable se composant d'un certains nombres d'autres variables (chaînes de caractères, numériques,...), rangées en mémoire les unes à la suite des autres. Le tableau est dit unidimensionnelle quand la variable élément de tableau n'est pas elle même un tableau. Dans le cas de tableaux imbriqués on parle de tableau unidimensionnels.

Les termes matrice, vecteur ou table sont équivalents à tableau.

Les tableaux unidimensionnels

Vous pouvez définir un tableau unidimensionnel avec la syntaxe suivante:**tab[index]=variable**, l'index est un numérique (mais pas obligatoirement, voir les tableaux associatifs), la variable peut être soit un numérique, soit une chaîne de caractère. Il n'est pas nécessaire de déclarer un tableau, la valeur initiale des éléments est une chaîne vide ou zéro. Exemple de définition d'un tableau avec une boucle **for**.

```
var=1
for (i=1;i<=NF;i++)
  { mon-tab[i]=var++}
```

On dispose de la fonction delete pour supprimer un tableau (**delete tab**). Pour supprimer un élément de tableau on tapera **delete tab[index]**.

Les tableaux associatifs

Un tableau associatif est un tableau unidimensionnel, à ceci près que les index sont des chaînes de caractères. Exemple:

```
age["olivier"]=27
age["veronique"]=25
age["benjamin"]=5
age["veronique"]=3
for (nom in age)
 { print nom " a " age[nom] "ans"
 }
```

On a un tableau **age** avec une chaîne de caractères prénom comme index, on lui affecte comme éléments de tableau un numérique (age de la personne mentionnée dans le prénom). Dans la boucle **for** la variable **nom** est remplie successivement des chaînes de caractères de l'index (**olivier**, **veronique**, ...).

Les valeurs de l'index ne sont pas toujours triées.

Les tableaux multidimensionnels

awk n'est pas prévu pour gérer les tableaux multidimensionnels (tableaux imbriqués, ou à plusieurs index), néanmoins on peut simuler un tableau à deux dimensions de la manière suivante. On utilise pour cela la variable prédéfinie **SUBSEP** qui, rappelons le, contient le séparateur d'indiçage. Le principe repose sur la création de deux indices (**i**, **j**) qu'on va concaténer avec **SUBSEP (i:j)**.

```
SUBSEP=":"
i="A",j="B"
tab[i,j]="Coucou"
```

L'élément de tableau "**Coucou**" est donc indexé par la chaîne "**A:B**".

Next Previous Contents