**INTRODUCTION**

Un **système d’exploitation** (SE) représente l’ensemble des logiciels qui, associé à un type de matériel, permet d’utiliser les ressources matérielles. Il supervise l’interaction entre le système informatique et les utilisateurs.

En règle générale, un système d’exploitation satisfait aux deux principales fonctions principales suivantes :

**1 - Partager les ressources** :

Une ressource peut être le processeur (CPU), la mémoire centrale (MC), périphérique d’entrée-sortie (E/S).

**But** : accroître la disponibilité de l’ordinateur pour les utilisateurs et en même temps, optimiser l’utilisation des ressources.

**2 – Présenter une machine virtuelle à l’utilisateur.**

**But** : libérer les programmeurs de la complexité du matériel. Le système d’exploitation masque beaucoup d’aspects fastidieux : interruption, gestion de fichiers et autres tâches dites de bas niveau.

Les tâches d’un système d’exploitation :

* **Chargement de programmes**.
* **Gestion des ressources** :
  + Protection mémoire (non violation de l’espace mémoire), ordonnancement des différents travaux, allocation de ressources, gestion des interruptions, …
* **Gestion de l’espace disque**:
  + Organisation des supports, implantation physique des fichiers, gestion de la protection des fichiers, …
* **Gestion des utilisateurs** :

Connexion, communication Homme/Machine (langage de commande), fonctions d’administration (création d’un nouvel utilisateur, modification des droits d’accès, …) services utilitaires (formatage, sauvegarde etc…).

**PRESENTATION UNIX**

**1 - ) Histoire d’Unix.**

* Unix est né dans les laboratoires BELL-LABS en 1969. Ken Thompson venait de réaliser un système interactif pour des besoins internes.
* En 1975, écriture du système en langage C (D.Ritchie). Le langage C est sorti en 1973. Le fait que ce système a été conçu en langage évolué, a permis une portabilité aisée.
* A partir de 1977, une deuxième équipe va travailler en parallèle sur UNIX, l’Université de Berkeley produit différentes versions notées x.y BSD.
* En 1979, les BELL-LABS sortent la version 7 ( V7) qui est à l’origine de l’expansion du marché UNIX.
* En 1984, Berkeley propose sa version 4.2 BSD et ATT son système V. Ce dernier a été autorisé à commercialiser ses produits en 1982.
* En parallèle avec cette évolution, sont apparus de nombreux produits à partir de 1980 : Les Unix dérivés.
* Depuis quelques années, le monde des systèmes d’exploitation pour ordinateurs de type PC vit un phénomène tout à fait original : l’émérgence et l’expansion ultra-rapide du système Linux (UNIX pour PC). Linux a pris naissance en 1991 dans l’esprit d’un seul homme, Linus Torvald, alors étudiant à l’Université d’Helsinki. Depuis lors, plusieurs dizaines de millions de personnes l’utilisent, bénéficiant du développement de plusieurs milliers d’informaticiens. Linux est, et demeurera sans doute, libre et gratuit, le code source étant librement consultable et modifiable par qui le souhaite.

**2 - ) Description d’Unix.**

**Unix est un système multi-tâches** : Cela signifie que plusieurs programmes peuvent s’exécuter simultanément ; par exemple, vous pourriez rédiger une lettre sur un traitement de texte, tout en attendant qu’un programme d’image de synthèse termine son calcul, et ce en même temps qu’un CD diffuse une musique. Il n’y a pas de limite théorique au nombre d’applications qui peuvent ainsi fonctionner simultanément : les facteurs limiteurs sont en premier lieu la quantité de mémoire dont dispose l’ordinateur, et en second lieu, la puissance du processeur. On peut avoir une cinquantaine de programmes (on dit aussi « processus ») qui s ‘exécutent simultanément.

**Unix est un système multi-utilisateurs** : C’est-à-dire que tout est prévu pour que plusieurs personnes utilisent le système, éventuellement simultanément.. En pratique, chaque personne utilisant le système, dispose d’un compte, qui peut être vu comme une certaine zone qui lui est allouée, accessible par un nom et un mot de passe. Il existe un dispositif de droits permettant d’assurer l’intégrité des données de l’ensemble des utilisateurs ainsi que du système proprement dit.

**Unix est fortement « orienté réseau »** : il offre tous les outils et tous les mécanismes pour la mise en place de réseaux locaux, l’intégration dans les réseaux existants, la connexion à internet.

**3 –) Structure du système**.

Le système est composé :

* **d’un noyau (kernel**) : il est chargé de diverses tâches indispensables à la marche du système.
  + - organise et gère les données du système.
    - veille à la protection du système et à l’accès aux informations.
    - véhicule l’information entre différents éléments du système.
    - supervise le fonctionnement multi-utilisateurs du système en planifiant l’exploitation du processeur entre les différents programmes exécutés.
    - assure la gestion de la mémoire centrale en l’attribuant aux différents programmes
    - enregistre l’activité du système.
* **d’un interpréteur de commandes** (appelé le **shell**), acceptant et exécutant les commandes des utilisateurs. Un shell est à la fois un langage de commandes et un langage de programmation interprété, permettant de générer de nouvelles commandes ou procédures cataloguées (scripts). Le shell dialogue avec les utilisateurs afin de recevoir leur requêtes, d’effectuer une première évaluation des paramètres et d’appeler les programmes utilitaires en fonction des demandes.

- **de programmes utilitaires** : éditeurs de textes, compilateurs de langages, des outils généraux de développement (débogueurs, constructeur d’analyseurs lexicaux et syntaxiques), un système de messagerie, etc…

# PRESENTATION DE L’ARBORESCENCE UNIX

**Les répertoires de base**.

Il existe une certaine normalisation dans l'architecture des répertoires sous Unix. Il peut exister certaines variations entre les différents Unix, mais dans l'ensemble c'est assez bien respecté.

**/** Dit répertoire racine (« root » en anglais) par analogie, dit-on, aux racines d’un arbre, représentant l’arborescence des répertoires et fichiers qui va se développer à partir de ce point. C’est donc le point de départ de toute la hiérarchie du système de fichiers. Ce répertoire est automatiquement attaché par le système lors de son démarrage. L’ensemble de ce système de fichiers partant de « / » (ou « root ») peut résider sur différents supports permanents de stockage physique (plusieurs disques par exemple) éventuellement découpés en partitions logiques. On peut y adjoindre des espaces de stockage amovibles (disquettes, CD,...), voire d’autres espaces repérés et gérés par d’autres machines en réseau (voire d’autres formes encore qu’on laissera pour la suite...). Ce découpage est complètement transparent pour les utilisateurs du système de fichiers. Les différentes parties peuvent être connectées au démarrage du système ou à la demande, en cours d’utilisation.

**/etc** Contient les fichiers de configuration généraux, les commandes à exécuter au démarrage, et même le mode de démarrage du système. Ce répertoire contient les **fichiers de configuration du système** utilisés au démarrage de la machine (enfin du PC !), voire au fil de l’eau en cas de besoin (par exemple, la fstab est utilisée pour s’allouer en dynamique des ressources stockées sur des supports amovibles - disquette, CDROM,...).

Vous trouverez dans

**/etc/passwd** - la liste des utilisateurs du système, leur identifiant système, pas leur mot de passe en clair, rassurez vous. **/etc/group** - la liste des groupes du système, **/etc/fstab** - les définitions et paramètres permettant de s’allouer des fichiers. **/etc/hosts** traduction d’adresses IP en noms d’hôte, ... etc... Vous trouverez dans les sous-répertoires suivants /**etc/X11/** - les fichiers de configuration de Xwindow, **/etc/rc.d**/ - les scripts de démarrage du système, **/etc/init.d/** - les scripts de démarrage des différents services (deamons) du systèmes (apache, mysql...), **/etc/cron/** - les tâches à mettre automatiquement en machine à une périodicité donnée (daily, hourly, monthly, weekly), **/etc/skel/** - les fichiers à recopier dans le répertoire d’un utilisateur qui vient d’être créé. **/etc/sysconfig/** - les fichiers de configuration des périphériques. **/etc/opt/** - les fichiers de configuration des applications

**/bin** Contient un **ensemble de fichiers exécutables représentant les commandes** que l’on peut adresser au système. On y trouve aussi des « utilitaires » permettant de travailler sur ou avec ces commandes système. Ce sont tous des fichiers binaires résultant de la compilation de programmes. Ils sont directement exécutables par le système et nécessaires à celui-ci. Tous les utilisateurs peuvent utiliser les commandes de ce répertoire. Une commande est spécifiée par son « nom » auquel est éventuellement associé un certain nombre de paramètres ou d’options.

Exemples de commandes dans /bin bash : « utilitaire » permettant d’enchaîner des commandes contenues dans des fichiers qui les décrivent. C’est ce que l’on appelle le Shell. ls commande permettant de lister le contenu du catalogue courant.

**/sbin** Contient des commandes nécessaires lors du démarrage du système, mais réservées à l'administrateur du système, ou super-utilisateur.

**/home** Est réservé à l'hébergement des comptes utilisateurs

**/dev** Ensemble de **répertoires et de fichiers système décrivant les périphériques** : ce sont des vecteurs de communication entre le système et ses périphériques qu'il s'agisse du clavier, du disque dur, de la carte son etc … Vous trouverez dans **/ dev/hda** - le premier disque dur IDE du système, **/dev/hda1** - la première partition de ce disque. **/dev/cdrom** - le lecteur de CD-ROM. **/dev/usb/scan**ner0 - le premier scanner USB. **/dev/console** - la console du système.

**/proc** Ce répertoire représente **le point de montage du pseudo système de fichiers du noyau**. Ce dernier contient des fichiers permettant d’accéder aux informations sur le matériel, la configuration du noyau et sur les processus en cours d’exécution

On retrouvera donc un répertoire par processus actif

Chacun porte le numéro du processus (PID) il est constitué des fichiers suivants : **cmdline** contient la ligne de commande qui a créé le processus, **status** contient des informations sur l’état du processus (en attente, en exécution, propriétaire... ), **exe** est un lien vers le fichier exécutable utilisé par le processus... Dans le répertoire /proc, on retrouve des fichiers contenants des informations générales sur le système. Comme :

uptime donnant le temps de fonctionnement du système, stat donnant diverses statistiques sur l’utilisation des ressources du système (CPU, mémoire... ), meminfo donnant un récapitulatif de l’utilisation de la mémoire. cpuinfo donnant une description des CPU du système...

**/boot** : Sont stockés ici les fichiers de démarrage du système .Contient un **ensemble de fichiers « Noyau » et fichiers de paramétrage nécessaires au démarrage du système**. Ces fichiers sont normalement en lecture seule.

**/mnt** : C'est en général à cet endroit qu'on accède aux autres systèmes de fichiers. C’est le catalogue définissant les « **points de montage** » des systèmes de fichiers non permanents (CD-ROM, disquettes... ) Ce répertoire contient autant de sous-répertoires que de systèmes de périphériques amovibles disponibles sur votre configuration.

**/media** : Certaines distributions montent les périphériques amovibles à cet endroit

**/lib** : Ce sont des bibliothèques utilisées par divers programme (C'est l'équivalent des DLL Windows). Par exemple, libjpeg.so permet à tous les programme de lire et écrire des fichiers JPEG

**/tmp** Contient les fichiers temporaires qui peuvent être détruits à chaque démarrage de la machine.

**/usr** Répertoire à usage multiples, dont les principaux sont :

**/usr/bin** Commandes utilisables par tous les utilisateurs et non nécessaires lors du démarrage du système.

**/usr/sbin** Commandes réservées au super-utilisateur, et non nécessaires lors du démarrage du système.

**/usr/include** Fichier d'en-tête des languages du système UNIX

**/usr/lib** Bibliothèque de fonctions utilisées par des langages de programmation et des macros pour le traitement de texte.

**/usr/man** Contient les manuels

**/usr/doc** Contient de nombreuse documentations et sources d'informations.

**/usr/X11R6** Contient tous les fichiers se rapportant à la couche graphique X-Windows

**/usr/local** Répertoire où sont stockés les macros et produits installés par l'administrateur.

**/var** Contient des données mises à jour par différents programmes durant le fonctionnement du système.

**/var/lock** Fichier de blocage, pour interdire par exemple deux utilisations simultanées.

**/var/spool** Répertoires utilisés pour l'organisation du travail des imprimantes, de la messagerie électronique, etc…

**/var/log** Contient les fichiers logs provenant des différents point du système. Un fichier log est un fichier contenant des informations concernant le déroulement d'un programme. Souvent très utiles en cas de problèmes ou de dysfonctionnement.

**LES UTILISATEURS**

# CREATION D'UN UTILISATEUR

**1 - ) UTILISATEUR**

Un utilisateur est identifié au niveau du système, par :

* son nom de **login**
* son mot de passe (commande **passwd**)
* son numéro d**'UID** (identificateur de l'utilisateur)
* son numéro de **GID** (identificateur de groupe)
* son répertoire d'accueil (**HOME DIRECTORY**)
* le programme lancé au moment de la connexion (en général, le **shell**)

1. **Le fichier /etc/passwd**

On déclare un utilisateur en créant une entrée dans le fichier **/etc/passwd**. Ce fichier peut être modifié :

* par n'importe quel éditeur de texte.
* par la commande **vipw**.
* géré à l'aide de procédures de commandes.

**/etc/passwd** contient une ligne par utilisateur. Cette ligne comporte sept champs séparés par le caractère : (deux points). La syntaxe de chaque ligne est la suivante :

**logname:password:UID:GID:libre:répertoire de connexion:shell**

* **logname** : nom de login

- il doit toujours commencer par un caractère alphanumérique.

- il est limité à 32 caractères ( Attention il peut être limité à 8 caractères sur certaines versions d'unix).

- il ne doit pas contenir d'espace.

- il doit être unique dans le système.

* **password** : mot de passe.

Il est crypté. Il peut être remplacé par le caractère "\*" ou le caractère "x" si le système est en **shadow password**. Dans ce cas, le mot de passe est crypté dans un autre fichier /etc/shadow

- **UID** : User Identifier.

Nombre entier unique dont se sert le système pour identifier l'utilisateur. L'UID O est réservé au super-utilisateur "**root**". Par convention, les numéros compris entre 0 et 99 sont réservés au système. Certaines distributions Linux commence les UID des utilisateur à 500. L'**UID** ne doit pas être supérieur à 59 999.

* **GID** : Group Identifier

Nombre entier qui détermine le groupe principal de l'utilisateur. Le groupe est déclaré dans le fichier /etc/group

* **libre** :

Zone libre dans laquelle on fait figurer des renseignements concernant l'utilisateur (Prénom Nom, Service, N° de téléphone). Attention, ces champs peuvent être utilisés par certaines commandes : le mail, finger etc…

* **répertoire de connexion** :

Appelé **HOME DIRECTORY**. Il renseigne la variable **HOME**. Il n'est pas nécessaire que le répertoire existe au moment de la création dans **/etc/passwd**. Par contre, il doit être créé pour que l'utilisateur puisse ouvrir une session.

* **shell** : interpréteur de commande.

C'est lui qui permettra de lancer des commandes et programmes à l'utilisateur.

## 2 - ) GROUPE

Un groupe est un ensemble d'utilisateurs ayant des points communs. Il est identifié par :

* + son nom
  + son numéro d'identification (**GID**)
  + la liste des utilisateurs qui le compose.

Si l'utilisateur appartient à un groupe, alors on modifie le fichier **/etc/group**, modifié par un éditeur de texte ou une procédure de commandes. Il comprend une ligne par groupe. Cette ligne est formée de quatre champ séparés par le caractère ":" (deux points)

**groupname::gid:logname1,logname2,…**

- **groupname** : nom du groupe

- **gid** : numéro d'identification du groupe.

- **logname1** liste des noms de login appartenant à ce groupe.

**Remarques** :

- Le deuxième champ n'est pas utilisé.

* La liste des noms de login n'est pas obligatoire. L'utilisateur appartiendra obligatoirement au groupe principal correspondant à son GID décrit dans son entrée du fichier **/etc/passwd**. Par contre un utilisateur peut appartenir à plusieurs groupes secondaires et dans ce cas là, il devra figurer dans la liste.

Lorsque les fichier **/etc/passwd** et **/etc/group** sont mis à jour, il faut exécuter les étapes suivantes :

* + création du répertoire de connexion : **mkdir nom\_de\_login**
  + création des variables d'environnement.

**3° ) Etude des commandes et autres fichiers se rapportant aux utilisateurs**

**a) La commande useradd.**

La commande **useradd** peut-être utilisée pour ajouter des utilisateurs dans votre système.

Avec la commande useradd, vous pouvez indiquer des paramètres :

exemple : **useradd –u 502 –g 300 –d /home/etu/jmm –s /bin/bash jmm**

Dans ce cas, l'utilisateur que nous venons de créer aura les caractéristiques suivantes :

UID : 502

GID : 300

Répertoire de connexion : /home/etu/jmm

shell : /bin/bash

Lorsqu'on ne précise pas de paramètres à la commande **useradd**, ils sont alors pris par défaut.

Les paramètres par défaut sont situés dans le fichier **/etc/default/useradd**. On peut connaître ces paramètres par défaut en tapant la commande useradd –D.

Si les paramètres par défaut ne sont pas ceux que vous souhaitiez, alors vous pouvez les modifier directement dans le fichier **/etc/default/useradd** ou par la commande **useradd –D**.

**Exemple** : - je veux que le groupe par défaut soit 100

- je veux que le mot de passe expire au bout de 60 jours

- je ne veux pas que le compte soit bloqué lors de l'expiration du mot de passe

- je veux que le shell par défaut soit /bin/bash

- je veux que le répertoire de connexion par défaut soit /home/etu

Pour effectuer ces changements, nous utilisons la commande :

**useradd –D –b /home/etu –g 100 –e 60 –f 0 –s /bin/sh**

Maintenant le fichier /etc/default/useradd sera le suivant :

GROUP=100

HOME=/home/etu

INACTIVE=0

EXPIRE=60

SHELL=/bin/bash

SKEL=/etc/skel

Par exemple pour ajouter un utilisateur fred, on peut utiliser la commande suivante :

useradd -m -u 510 –g 300 –c "Frederic Dupont" –s /bin/bash fred

cela créera l'entrée dans /etc/passwd :

fred:x:510:300:Frederic Dupont:/home/etu/fred:/bin/bash

Nous aurons aussi une entrée dans **/etc/shadow**

**b) le fichier /etc/shadow**

Exemple d'entrée dans **/etc/shadow**: Les paramètres par défaut sont définis dans le fichier **/etc/login.defs**

fred:!!:12368:0:99999:7:::

Le premier champ correspond au login.

Le deuxième champ correspond au mot de passe crypté. Attention, le mot de passe est initialisé par la commande **passwd**.

Le champ suivant correspond au nombre de jours écoulés entre le 1 er Janvier 1970 et la dernière modification du mot de passe.que le mot de passe est initialisé.

Le champ suivant correspond au nombre de jour minimum entre deux changements de mot de passe. ( 0 indique que le mot de passe peut être changé à tout moment )

Le champ suivant correspond au nombre de jour maximum qu'un mot de passe soit validé (99999 correspond à un grand nombre d'année). Si ce champ est inférieur à la valeur du champ précédent, l'utilisateur ne pourra pas changer son mot de passe.

Le champ suivant correspond au nombre de jour pour avertir l'utilisateur de l'expiration de son mot de passe (7 correspond à une semaine pleine)

Le champ suivant correspond au nombre de jour avant que le compte soit bloqué suite à l'expiration du mot de passe

Le nombre de jours depuis le 1 janvier 1970, qu'un compte a été neutralisé.

Le dernier champ n'est pas utilisé.

**c) le fichier /etc/login.defs**

Le fichier **login.defs** contient des paramètres concernant les mots de passe :

PASS\_MAX\_DAYS 99999 : nombre de jour maximum de validation d'un mot de passe

PASS\_MIN\_DAYS 0 : nombre de jour minimum de validation d'un mot de passe

PASS\_MIN\_LEN 5 : longueur minimale d'un mot de passe

PASS\_WARN\_AGE 7 : nombre de jours pour avertir l'expiration d'un mot de passe

UID\_MIN 500 : valeur minimale du GID pour créer un groupe par défaut

UID\_MAX 6000 : valeur du GID maximum

**d) la commande chage**

Cette commande permet de configurer le fichier /etc/shadow. C'est elle qui permet de visualiser avec l'option –l, ou de modifier les attributs qui traitent de la pérennité du mot de passe.

Exemple : fixer la date d'expiration du mot de passe d'un utilisateur:

# chage –E aaaa /mm/jj Nom\_utilisateur

Visualiser les attributs du mot de passe d'un utilisateur

# chage –l Nom\_utilisateur

**e) la commande pwconv**

Cette commande permet de passer en "shadow passwd". Elle crée le fichier /etc/shadow à partir de /etc/passwd. La commande **pwunconv** passe du mode *shadow* au mode non sécurisé, c'est-à-dire que les mots de passe apparaitront cryptés dans le fichier /etc/passwd.

**f) la commande usermod**

La commande **usermod** est utilisée pour modifier les informations relatives à un utilisateur. Les options sont les mêmes que pour **useradd**.

**g) la commande userdel**

La commande **userdel** supprime un compte. Si **userdel** est utilisée avec l'option **–r**, alors tous les fichiers et sous-répertoires du HOME DIRECTORY seront supprimés. Les fichiers et répertoires situés en dehors du répertoire d'accueil, devront être recherchés et effacés manuellement.

**h) Programmes de vérification de la structure**.

* **pwck** : cette commande est fournie pour vérifier la cohérence des fichiers /etc/passwd et /etc/shadow. Elle vérifie chaque nom d'utilisateur ainsi que les points suivants :

Le nombre correct de champs

Nom unique

Nom et groupe valide

Groupe primaire valide

Répertoire d'accueil valide

Shell valide

Elle prévient aussi lorsqu'un compte ne possède pas de mot de passe.

* **grpck** : cette commande est fournie pour vérifier la cohérence des fichiers /etc/group. Elle effectue les vérifications suivantes :

Le nombre de champs corrects

L'unicité du nom de groupe

La validité de la liste des membres et des administrateurs.

**La gestion des utilisateurs se fait par l'administrateur du système**.

**4° ) Etude de l'environnement des utilisateurs.**

Résultat de la commande **env**, en italiques : commentaires

**USERNAME**=stagex *<-- Nom d'utilisateur*  
**HISTSIZE**=1000     *<-- Nombre de commandes enregistrées dans l'historique*  
**HOSTNAME**=machine1.domain.fr **LOGNAME**=stagex  *<-- redondant avec username mais demandé par certaines commandes*  
**MAIL**=/var/spool/mail/stagex  *<-- Répertoire du courrier entrant*  
**TERM**=vt100   *<-- Modèle de terminal*  
**HOSTTYPE**=i386  *<-- Type processeur*

*Ordre de recherche des commandes dans les répertoires*  
**PATH**=/opt/kde/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/dislin/bin:/usr/local/lam/bin:/home/phys/enseign/epelboin/bin  
**KDEDIR**=/opt/kde   *<-- Répertoire de configuration de l'environnement KDE*  
**HOME**=/home/phys/enseign/stagex  *<--  Répertoire home (défini dans* [*/etc/passwd)*](http://www.lmcp.jussieu.fr/impmc/Enseignement/ye/informatique/unix/documents/passwd.html)  
**SHELL**=/bin/bash   *<--  Shell )par défaut (défini dans* [*/etc/passwd*](http://www.lmcp.jussieu.fr/impmc/Enseignement/ye/informatique/unix/documents/passwd.html)*)*  
**PS1**=[\u@\h \W]\$   *<--  Prompt*  
**USER**=stagex   *<---  Encore une définition de l'utilisateur!*  
**BASH\_ENV**=/home/phys/enseign/stagex/.bashrc *<--  Fichier de configuration du shell*  
**OSTYPE**=Linux

## Script /etc/profile executé en début de session

*Ce script, rangé dans /etc/profile est celui par défaut, employé par tous les utilisateurs.*

# /etc/profile  
# System wide environment and startup programs  
# Functions and aliases go in /etc/bashrc  
PATH="$PATH:/usr/X11R6/bin"  
PS1="[\u@\h \W]\\$ "  
ulimit -c 1000000

*Positionnement du masque de droits pour les fichiers. On distingue le système (id <= 14) et pour les usagers "ordinaires" : rwx propriétaire, rx groupe, rien pour les autres*  
if [ `id -gn` = `id -un` -a `id -u` -gt 14 ]; then  
      umask 007  
else  
      umask 027  
fi  
USER=`id -un`  
LOGNAME=$USER  
MAIL="/var/spool/mail/$USER"  
HOSTNAME=`/bin/hostname`  
HISTSIZE=1000  
export PATH PS1 HOSTNAME HISTSIZE USER LOGNAME MAIL

*S'il existe un script complémentaire dans /etc/profile.d on l'exécute (source) dans le contexte de l'utilisateur*

for i in /etc/profile.d/\*.sh ; do  
      if [ -x $i ]; then  
            . $i  
      fi  
done  
unset i

## Exemple script dans profile.d

*Il existe un script pour initialiser l'environnement kde : kde. sh*

# KDE initialization script (sh)  
if [ -z "$KDEDIR"  -o  "$KDEDIR" != "/opt/kde" ] ; then  
        KDEDIR="/opt/kde"  
        PATH="$KDEDIR/bin:$PATH"  
fi  
export  KDEDIR PATH

## Initialisation du shell bash

*Script général de démarrage de bash, exécuté avant celui propre à l'utilisateur*

/etc/bashrc  
# System wide functions and aliases  
# Environment stuff goes in /etc/profile  
# For some unknown reason bash refuses to inherit  
# PS1 in some circumstances that I can't figure out.  
# Putting PS1 here ensures that it gets loaded every time.

PS1="[\u@\h \W]\\$ "  
alias which="type -path"  
export PATH=$PATH:/usr/local/lam/bin

## Profil particulier utilisateur

*Le profil est séparé en deux parties :*

1. *la définition des alias et l’exécution du script de début de shell*
2. *la définition de l’environnement et du nom du script de début de shell, qui sera lancé, s’il existe, à l’ouverture de chaque shell.*

*Script .bashrc : lance le script général puis les extensions propres à chacun*

# User specific aliases and functions  
# Source global definitions

if [ -f /etc/bashrc ]; then  
        . /etc/bashrc  
fi

alias rm "rm -i"

*Script  .bashrc\_profile : entièrement à définir par l'utilisateur*

# User specific environment and startup programs

PATH=$PATH:$HOME/bin  
BASH\_ENV=$HOME/.bashrc  
USERNAME=""  
export USERNAME BASH\_ENV PATH

#### GESTION DE L’AUTHENTIFICATION D’UN UTILISATEUR AVEC

**1°) Introduction**.

PAM est l’acronyme de Pluggable Authentification Modules (modules d’authentification enfichables).

PAM est le système d’authentification utilisé en standard sous Linux. PAM est un système permettant de gérer individuellement l’authentification pour les applications. Pour cela, il suffit de configurer dans un fichier du même nom que le service ayant besoin d’authentification dans le répertoire /etc/pam.d.

**2°) Format des lignes de configuration des fichiers de /etc/pam.d**

Le format de chaque ligne des fichiers de configuration dans les fichiers de /etc/pam.d est :

module-type control-flag module-path arguments

1. **Type de modules** : module-type

* **account** : ce module est chargé de vérifier si l’identification est autorisée (expiration de compte, plage horaire de connexion, …)
* **auth** : ce module est utilisé pour vérifier si l’utilisateur qui veut se connecter est bien celui qu’il prétend être. Ceci passe par un moyen d’authentification comme :

un mot de passe

une phrase de passe

une accès matériel type carte à puce

un accès de type biométrique

* **password** : ce module permet à l’utilisateur de modifier ce qui lui sert pour l’authentification (mot de passe, carte à puce etc…)
* **session** : ce module est utilisé à l’intérieur de la session (après l’identification). Il permet de contrôler, par exemple, l’accès aux répertoires, aux boites mail etc…

1. **Importance du module**: control-flag

* **required**: indique que le succès du module est nécessaire pour autoriser le service. Les autres lignes du fichier sont également lues et ce n’est qu’après que le module est accordé ou non.
* **requisite**: indique que le succès du module est nécessaire pour autoriser le service. Les tests sont stoppés dès le premier échec, ceci permettant de mettre en place une sécurité accrue dans les milieux « hostiles ». Si par exemple, on veut s’assurer que seul tty1 peut être utilisé par le root, on utilise ce drapeau. Ainsi, un utilisateur sur une autre console, n’aura même pas l’occasion d’essayer un mot de passe.
* **sufficient** : indique que le succès suffit à arrêter d’exécuter les modules du même type suivant celui-ci.
* **optional** : indique de ne pas tenir compte du résultat de ce module. C’est uniquement en dernier recours que son ‘avis’ sera pris en compte, lorsque, par exemple, les autres modules renvoie *PAM\_IGNORE*.

#### Le chemin et le nom du module : module-path

#### Indique le nom du module (.so) relatif à /lib/security ou /usr/lib/security si ce nom ne commence pas par un /.

#### 3°) Quelques modules usuels

**pam\_cracklib :** Permet d'accepter ou de rejeter un mot de passe, si celui-ci se trouve dans un dictionnaire. Il permet aussi de vérifier que vous ne réutilisez pas le même mot de passe. Vous pouvez le faire suivre de retry=n (le nombre de tentatives) minlen=n (la longueur imposée) difok=n (nombre de caractères qui sont dans le vieux mot de passe et que l'on ne peut pas retrouver dans le nouveau).

**pam\_env :**  Permet de spécifier des variables d'environnements spécifiées dans /etc/security/pam\_env.conf à tout utilisateur qui se connecte sur la machine.

**pam\_unix :** Module de base. Gère à la mode unix la politique d'authentification. Il peut être avec les quatre types de modules : account (établi la validité utilisateur/mot de passe et peut forcer la modification de celui là), auth (compare avec la base le mot de passe), password (la politique de changement du mot de passe), session (pour loguer les connexions).  
Vous pouvez associer quelques options dont : nullock pour autoriser un mot de passe vide, md5 pour le type de cryptage, debug pour loguer les informations à syslog, remember=n pour ce souvenir des n derniers mots de passe utilisés.

**pam\_pwdb :** module de base, qui a les mêmes options que pam\_unix.

**pam\_time :** autorise un accès par heure. La configuration se faisant dans le fichier /etc/security/time.conf.

**pam\_wheel :** permet de limiter l'accès à root via la commande su qu'aux seuls membres du groupe wheel. On peut changer le nom du groupe par défaut avec l'option group=mon\_group.

**pam\_limits :**  Permet de limiter les ressources mis à la disposition d'un utilisateur. Il faut alors configurer le fichier /etc/security/limits.conf. Avec l’argument **conf=filename** indique le chemin du fichier limits.conf s’il n’est pas dans /etc/security

**pam\_nologin :** permet de désactiver les comptes. Il faut alors créer le fichier /etc/nologin et alors il n'y a plus que root qui puisse se connecter.

**pam\_access :** Ce module permet de contrôler les utilisateurs par nom, machine, domaine, adresse IP, terminal. Vous devez alors configurer le fichier /etc/security/access.conf

**pam\_deny :** comme sont nom l'indique. Vous pouvez (devez !) l'utiliser dans /etc/pam.d/other pour auth, account, password et session avec required. Si dans le répertoire /etc/pam.d vous avez des noms d'applications que vous n'utilisez pas, vous pouvez renommer ces fichiers avec un autre nom au cas..! Si quelqu'un cherche à utiliser l'application le other sera alors utilisé par défaut.

**pam\_securetty :** Vérifie que le compte root a la possibilité de se connecter sur cette console. Pour cela il faut qu'elle soit indiquée dans le fichier /etc/securetty.

**pam\_warm :** log les informations à syslog

**pam\_console :** permet de spécifier les autorisations d'accès à la console. Il faut alors configurer /etc/security/console.perms.

**pam\_stack :**  Indique un fichier de configuration PAM à inclure au fichier en cours (modtype : tous). Avec l’argument **service=nom\_service** Nom du service du fichier de configuration PAM à inclure. Généralement suivi de service=system-auth, permet de renvoyer sur system-auth.

**pam\_ldap :** permet d'effectuer l'authentification sur une base ldap. Ce module demande une documentation à lui  tout seul.

**pam\_listfile.so** : Indique une liste d’utilisateurs autorisés ou interdits à se connecter (modtype : auth) à partir d’un fichier

Avec l’argument **onerr=succeed|fail** (obligatoire) indique si en cas d’erreur (fichier inexistant), on renvoie succès ou echec.

Avec l’argument **sense=allow|deny** (obligatoire) indique si la liste est autorisante ou interdisante.

Avec l’argument **file=filename** (obligatoire) indique si le fichier contenant la liste d’utilisateur (un par ligne)

**pam\_motd.so** : Affiche le contenu de /etc/motd à la connexion (modtype : session)

Avec l’argument **motd=filename** (facultatif) si le fichier motd n’est pas /etc/motd par défaut.

#### GESTION DISQUES ET FICHIERS SYSTEME DE GESTION DE FICHIERS

**1 - ) FICHIERS**

Le système de gestion de fichiers Unix est un **système hiérarchisé** dans lequel on distingue trois types de fichiers :

**A -) Fichiers ordinaires**.

Fondamentalement, un fichier est une suite de caractères (octets) qui constitue un ensemble cohérent d'informations. Le fichier est l'élément fondamental d'un système d'exploitation : les instructions nécessaires au fonctionnement de ce système, sa configuration etc… toutes ces informations sont stockées dans les fichiers. Chaque fichier est repéré par son nom qui doit respecter certaines règles. Enfin, les fichiers sont eux-mêmes stockés physiquement sur des supports comme les disquettes, disques, CD-ROM, etc…

Sous Unix, les noms de fichiers sont limités à 255 caractères et ne doivent en principe comporter que des caractères alphanumériques (c'est-à-dire que des lettres et des chiffres), ainsi que les symboles . - \_ + %

Il est possible d'insérer un ou plusieurs espaces ou d'autres caractères spéciaux mais attention, vous risquez d'avoir des problèmes lors de la manipulation de ces fichiers. Le système respecte la casse cela veut dire que le *Fichier.txt* est différent du fichier *fichier.txt*

Les fichiers (ou répertoires) dont le nom commence par un point ("."), sont des fichiers (ou répertoires) cachés. Ils ne seront listés par la commande **ls** sans paramètres.

**B -) Répertoires.**

Les répertoires sont des fichiers un peu particuliers, qui contiennent d'autres fichiers et répertoires (appelés sous-répertoires) plus précisément qui contiennent les références et descriptions d'autres fichiers et répertoires. Les répertoires assurent la correspondance entre un nom de fichier et sa localisation dans l'arborescence (c'est un catalogue de fichiers et sous-répertoires).

Au moment de la création d'un répertoire, deux entrées sont automatiquement crées :

"**.**" représentant le répertoire courant.

"**..**" représentant le répertoire père.

**C -) Fichiers périphériques.**

Ils correspondent à des ressources (ex : terminaux, imprimantes, disque physique). Ils sont traités comme des fichiers sur disque ordinaires mais les opérations de lecture et d'écriture sur ces fichiers activent les dispositifs physiques associés. Une caractéristique de ces fichiers, leur taille est nulle.

Exemple : /dev/tty01 représente le terminal relié à l'ordinateur via la ligne de communication "tty01"

**D - ) Autres fichiers.**

Fichier tubes nommés (communication entre processus, fichier **pipe**), les **sockets** (communication bidirectionnelle entre processus), les **liens symboliques** (fichier dont le contenu est interprété comme un nom de fichier)

**2 - ) ARBORESCENCE**

Nous avons une **structure arborescente** avec des nœuds (les répertoires) et des feuilles (fichiers ordinaires ou spéciaux). La racine **(/)** ou **root**, est la base de toute structure. Le nom complet d'un fichier (ou chemin d'accès ou **pathname**) est constitué de la suite des nœuds empruntés, séparés par le caractère **/** (slash).

Notion de catalogue de travail (**working directory**) : à tout instant, on se trouve dans un répertoire désigné par le caractère "**.**" (point) : c'est le répertoire courant. Le chemin où l'on se trouve s'obtient par la commande **pwd**.

On se déplace dans l'arborescence par la commande **cd** selon

un chemin d'accès absolu (à partir de la racine / (exemple : cd /home/etu) )

ou relatif (par rapport à l'endroit où l'on se trouve).

**Exemple** :

Pour remonter dans l'arborescence cd ..

Pour accéder à des répertoires collatéraux cd ../etu

Pour retourner à son home directory cd

**3 - ) DESCRIPTION DES FICHIERS.**

Tout fichier ou répertoire correspond à une entrée dans une table contenant l’ensemble de ses attributs. Une telle entrée est appelée **i-node** (index node).

Identification d’un fichier=

Identification de la table dans laquelle sont enregistrées ses caractéristiques

+

Indice dans cette table de l’entrée contenant ses caractéristiques

A chaque **i-node** est associé les informations suivantes :

* Le nom du propriétaire modifiable par la commande **chown**
* Le nom du groupe modifiable par la commande **chgrp**
* Le nombre total de liens sur ce fichier
* La taille du fichier
* Les dates de création, dernière modification, dernier accès en lecture.
* Type de fichier :
  + - **d** : répertoire
    - **-** : fichier normal
    - **b** : fichier bloc
    - **c** : fichier caractère
    - **l** : fichier symbolique
* Les droits d’accès au niveau du propriétaire, du groupe et des autres.

On obtient la liste du contenu d’un répertoire par la commande **ls**. Cette commande possède beaucoup d’options :

Exemples : **ls –lsi** donne un maximum d’informations ( en particulier l’inode par "**i**")

**ls –a** affiche les fichiers ou répertoire cachés. Un fichier ou répertoire caché ont leur nom qui commence par un . (point).

**4 - ) DROITS D’ACCES.**

*Le super-utilisateur a tous les droits***.**

Sous Unix, tout utilisateur possède un numéro de compte et appartient à un groupe particulier.

**4-1) Les différents droits.**

**Pour un fichier donné, les utilisateurs du système peuvent être classés en trois catégories :**

* le propriétaire (en général, le créateur) symbolisé par la lettre **u** comme **User** .
* les membres du même groupe  que le propriétaire symbolisé par la lettre **g** comme **Group**.
* Les autres utilisateurs symbolisé par lalettre **o** comme **Other**.

**Trois types d’opérations élémentaires sont contrôlés par le système** :

Pour les fichiers :

* droit de lecture (symbolisé par la lettre **r** comme **Read**)
* droit d’écriture (symbolisé par la lettre **w** comme **Write**)
* droit d’exécution (symbolisé par la lettre **x** comme **eXecute**)

Pour les répertoires :

* **r** : droit de lire les noms référencés dans ce répertoire.
* **w** : droit d’écrire dans ce répertoire et en particulier de détruire un fichier
* **x** : permission de traverser ce répertoire ou de s’y positionner.

A tout fichier (au sens large : cela s‘applique aussi aux répertoires) est finalement associé un ensemble de droits d’accès caractérisé par 9 lettres (3\*3).

**rwx rwx rwx**

**u g o**

Lorsqu’ un droit est retiré, la lettre est remplacé par un – (tiret). Ce changement de protection se fait par la commande **chmod**.

Pour assurer la protection, le système utilise donc 9 bits de **l’i-node** du fichier. De plus 3 autres bits ont un rôle particulier.

Propriétaire du fichier Groupe du fichier Tous les autres

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

Exécution

ID Sticky

Propriétaire Ecriture

ID groupe

Lecture

Attention, le droit "**w**" est très étendu, et même dangereux, quand il est accordé à un groupe, car un membre du groupe peut supprimer des fichiers

# Cas particulier :

# 

# En dehors des droits r w x, il existe d’autres attributs que l’on peut donner à un fichier.

# a -) Le set-uid  bit (s) :

Sa présence permet à un fichier exécutable **de s'exécuter sous l'identité et donc les droits de son** **propriétaire**, à la place de l'utilisateur actuel qui l'exécute.

Il s'agit d'un dispositif de sécurité essentiel qui autorise un utilisateur quelconque (par rapport à la commande) à bénéficier de droits plus étendus que les siens (souvent ceux de **root**), pour exécuter la commande, agir sur d'autres fichiers indispensables, juste le temps et sous le contrôle de l'exécution de la commande, sans qu'il soit nécessaire d'attribuer ces droits en permanence sur les fichiers.

Ce droit est noté symboliquement "**s"** et se positionne à la place du "**x**" du propriétaire (mais sans écraser le droit "**x"**). Si le droit "**x"** n'est pas accordé, il apparaîtra la lettre "**S"**.

Sa valeur octale est 4000

Exemple significatif

Examiner les droits du fichier exécutable **/usr/bin/passwd**, qui permet de changer un mot de passe et le comparer à ceux du fichier **/etc/shadow** qui contient les mots de passe cryptés.

La commande : **ll /etc/shadow** , donne le résultat suivant :

**-r-------- root root shadow**

**ll –l /usr/bin/passwd**

**-r-sr-xr-x root root passwd**

Comme le droit "**x"** est accordé à tous, chacun peut donc exécuter la commande **passwd**, mais personne ne possède le droit le droit d'écriture dans le fichier **/etc/shadow** qui doit le stocker.

Le positionnement du **SUID** permet d'agir en tant que **root** lors de la demande d'accès au fichier et comme **root** a tous les droits, il est alors possible de mettre à jour ce fichier des mots de passe.

**Comment connaître les commandes qui offrent cette permission SUID ?**

Voici plusieurs façons :

**cd /usr/bin**

# grep filtre les lignes produites par ls en utilisant l'expression rationnelle **^…s**

**ls –l | grep "^…s"**

# pour afficher tous les fichiers possédant le **SUID**

**cd /**

**ls –lR | grep "^…s"**

# recherché parmi les fichiers ordinaires ceux qui ont au moins le droit "**s**"

**find / -type f –perm +4000**

**b - ) Le set-gid bit (s)** :

Pour un fichier exécutable, il fonctionne de la même façon que le **SUID**, mais transposé aux membres du groupe.

Exemple :

* Examiner les droits symboliques de la commande d'impression **/usr/bin/lpr**
* Quelle est sa valeur octale ?
* Si une imprimante a été installée, un répertoire lp a été créé dans **/var/spool/lpd**. Or la commande **lpr** écrit dans ce répertoire. Comment un utilisateur quelconque peut-il alors écrire le fichier d'impression ?

Positionné sur un répertoire, ce droit modifie le groupe propriétaire d'un fichier créé dans ce répertoire.

Un fichier créé dans un tel répertoire, verra son groupe propriétaire modifié :

Ce ne sera plus le groupe primaire du propriétaire qui l'a créé (règle habituelle), mais à la place, le groupe propriétaire du répertoire lui-même.

Autrement dit, ce droit **"s"** posé sur un répertoire, met en place un mécanisme d'héritage de groupe, de répertoire conteneur à fichiers contenus.

Notation symbolique "**s"**, mis à la place du "**x"** du groupe, valeur octale 2000

**c - ) Le sticky bit (t)** :

Ce droit spécial a surtout un rôle important sur les répertoires. Il réglemente le droit "**w"** sur le répertoire, en interdisant à un utilisateur quelconque de supprimer un fichier dont il n'est pas le propriétaire.

Ce droit, noté symboliquement "**t"**, occupe par convention la place du droit "**x"** sur la catégorie **other** de ce répertoire, mais bien entendu il ne supprime pas le droit d'accès "**x"** (s'il est accordé).

Si le droit "**x"** n'est pas accordé, c'est la lettre "**T**" qui apparaît à la place de "**t"**.

Exemple : **chmod +t rep**

**d … … ...t rep** si le répertoire a le droit x pour tous

**d … … ..T rep** sinon

**4 – 2 ) Définition des permissions d’accès**.

De façon générale, l'utilisateur qui crée un fichier en devient le propriétaire, et le groupe auquel l'utilisateur appartient (au moment de la création) devient le groupe du fichier.

Remarques:

Les droits accordés au propriétaire, au groupe et aux autres dépendent du processus qui a créé le fichier et du **masque des droits**.

L'administrateur peut être amené à effectuer les changements de propriété (par exemple pour permettre un travail en groupe) et des changements de droits sur des ensembles de fichiers et répertoires, les étendre ou les restreindre.

**root** n'est pas soumis à ces restrictions, il a le pouvoir absolu sur le système de fichiers.

**a - ) la commande chown**

**chown [-option] ident\_utilisateur réf …**

Permet de changer le propriétaire d'un ou de plusieurs fichiers (ou répertoires). Seul, le propriétaire d'un fichier ou l'utilisateur **root** peut changer le propriétaire d'un fichier. Option "**–R"** permet un changement récursif de propriétaire (sur toute l'arborescence de racine donnée)

**b - ) la commande chgrp**

**chgrp [-option] ident\_group réf …**

Permet de changer le groupe propriétaire d'un ou plusieurs fichiers (ou répertoires). Seul, le propriétaire du fichier ou l'utilisateur **root** peut changer le groupe propriétaire d'un fichier. Option "**–R"** permet un changement récursif de groupe (sur toute l'arborescence de racine donnée)

# c - ) la commande chmod.

# 

# Syntaxe : chmod argument réf …

Cette commande modifie les droits d’accès aux fichiers (ou répertoires) référencés . Attention, **seul le** **propriétaire d’un fichier ou** **root** peut changer les droits d’accès du fichier**.**

La commande **chmod** peut s'écrire de plusieurs façons équivalentes, sur le modèle :

**chmod *droits fichiers***

Le paramètre "*droits*" permet de calculer les nouveaux droits d'accès.

Les droits peuvent être modifiés de façon relative, par ajout (symbole +) ou retrait (symbole -) par rapport aux droits existants, ou bien de façon absolue, en fixant les nouveaux droit qui remplacent les anciens (symbole =)

# Argument :

# - mode absolu

# 4000  : positionnement du set-uid bit ;

# 2000  : positionnement du set-gid bit ;

# 1000  : positionnement du sticky bit ;

# 400  : lisible par le propriétaire ;

# 040  : lisible par les membres du groupe propriétaire ;

# 004  : lisible par les autres utilisateurs ;

# 200  : modifiable par le propriétaire ;

# 020  : modifiable par les groupe propriétaire ;

# 002  : modifiable par les autres utilisateurs ;

# 100  : exécutable par le propriétaire ;

# 010  : exécutable par le groupe ;

# 001 : exécutable par les autres utilisateurs ;

# La combinaison des modes se fait en additionnant les valeurs.

# Exemple : 761 : lecture (4), écriture (2), exécution (1) pour le propriétaire

# lecture (4), écriture (2) pour le groupe

# exécution (1) pour les autres

**Ajout, retrait ou fixation des permissions**.

Pour chaque fichier ou répertoire) , on désigne par :

- **u, g, o**, les 3 catégories d'utilisateurs (**user, group,other**) et de plus par "**a**" (**all**) tous les utilisateurs - **r, w, x , s** **(set\_uid et set\_gid), t** attributs de chaque fichier, pour chaque catégorie d'utilisateurs.

- **+, -, =** l'action d'ajouter, de retirer, ou de fixer un droit, qui s'applique à chaque catégorie séparément.

- Les changements sur le modèle "à quelle(s) catégorie(s), quelle action, quel(s) droit(s)" sont notés symboliquement :

**[u g o a ] [+ - =] [ r w x]**

**Notation relative** (au droit existant)

Exemple : **chmod ug+x /home/toto/fichier** : "ajoute le droit d'exécution au propriétaire du fichier"

**chmod go-rwx /home/toto/fichier :** "enlève tous les droits d'accès à tous les utilisateurs, sauf au propriétaire"

**Notation absolue**.

Pour chaque groupe, elle permet de fixer les nouveaux droits qui remplacent les anciens. Si une catégorie n'est pas présente, ses anciens droits s'appliquent.

Exemple : **chmod u=rwx,g=rw,o=r /home/toto/fichier** remplace les permissions précédentes des fichiers en les fixant **à –rwxrw-r—**

Attention, aucun espace dans la liste des droits, même après les éventuelles virgules.

**chmod u=rwx,g=r /home/toto/fichier** fixe les permissions à **–rwxr--???** En ne changeant pas les droits de **other**.

**d - ) commande umask**

Syntaxe : **umask [mode]**

Rappelons les règles simples de propriété qui s'appliquent à la création d'un fichier ou d'un répertoire :

* son propriétaire est l'utilisateur qui l'a créé.
* Son groupe est le groupe primaire de ce même utilisateur.

Mais quelles sont les permissions accordées par défaut à l'utilisateur propriétaire, au groupe propriétaire et tous les autres ?

Les permissions maximales accordées par un fichier et un répertoire sont **666 (-rw-rw-rw-) et 777 (-rwxrwxrwx)**.

On peut restreindre ces permissions lors de sa création. C'est le rôle de la commande **umask** de fixer les permissions masquées, autrement dit les droits non accordés aux fichiers et répertoires lors de leur création.

Remarquez que le premier digit, c’est-à-dire les trois premiers bits (SUID,SGID,sticky-bit) n’est jamais manipulé, et que l’accès de type execute n’est pas pris en compte dans le maque pour des raisons de sécurité.

Exemple de calcul de permissions effectives, affectées lors de la création d'un répertoire, par un utilisateur dont le masque est **027**

1. = 111 111 111 permissions maxi = rwx rwx rwx

- 027 = 000 010 111 masque de protection

= 750 = 111 101 000 permissions effectives = rwx r-x ---

* **umask** affiche le masque de l'utilisateur actif
* **umask –S** affiche les permissions correspondantes au masque, sous forme symbolique
* **umask** masque fixe les permissions ultérieures de création de fichiers de l'utilisateur actif, conformément à masque, en notation octale. Attention, le changement ne s'applique qu'à la présente session.

Pour la rendre permanente, on peut intervenir sur un fichier **profile** :

* dans le fichier profile général **/etc/profile**, on peut modifier la règle :

**if [ $UID == 0 ] ; then umask 022; else umask 077 ; fi**

L'utilisateur peut agir au niveau de son propre **.profile**

**e -) la commande chattr et lsattr**

Les [systèmes de fichiers](http://www.lea-linux.org/cached/action=edit/index/Syst%C3%A8me_de_fichiers.html) *ext2* et *ext3* offrent la possibilité de donner à des fichiers (fichiers réels ou répertoires) des **attributs (***extended attributs*) qui s'ajoutent aux [permissions classiques POSIX.1](http://www.lea-linux.org/cached/index/Permissions.html), éventuellement complétées par les [permissions étendues ACL](http://www.lea-linux.org/cached/index/ACL.html).

ll existe deux commandes pour accéder aux attributs étendus : *lsattr* (*list attributes*, « lister les attributs ») *chattr* (*change attributes*, « changer les attributs »). Toutes deux sont incluses dans le paquetage *e2fsprogs*.

**Commande chattr**  Cette commande s'utilise quasiment comme un *chmod* simplifié : en effet, les attributs ne peuvent être indiqués que sous forme symbolique (dont les codes sont donnés ci-dessus) et non au moyen d'un éventuel nombre octal. Ainsi, pour ajouter ou donner un attribut, on utilise la commande suivie de + et du code ; par exemple, rendre un fichier « immuable » se fait avec la commande (en root) : chattr +i nom\_fichier

## Types d'attributs

Selon la norme POSIX.1, tout fichier possède les permissions en lecture, écriture et exécution. On peut leur adjoindre des attributs supplémentaires, qui s'imposent à tous les utilisateurs (dont root, qui peut cependant annuler ces attributs) quand ils modifient les actions qu'on peut avoir sur eux. Certains attributs sont encore expérimentaux (octobre 2005) et nécessitent de patcher le noyau pour qu'ils aient un effet réel.

**append only (a),** « ajouter seulement » ─ *réservé à root*

Protège le fichier contre la suppression de contenu : on ne peut que lui en ajouter.

**compressed (c),** « compressé » ─ *expérimental*

Le fichier est automatiquement compressé ─ si c'est utile, et avec un format de compression le plus efficace ─ quand il est stocké. La décompression lors de la lecture et la compression des données ajoutées au fichier par la suite se font à la volée, de manière transparente pour l'utilisateur (la commande ls -l, par exemple, montre la taille décompressée).

Cet attribut appliqué à un répertoire ne le compresse pas mais fait se compresser automatiquement tout nouveau fichier qu'on y crée.

**no dump (d),** « pas de sauvegarde »

Un fichier ainsi marqué sera ignoré par la commande de sauvegarde *dump* (qui ne fonctionne que pour *ext2* et *ext3*). Cet attribut est utile pour empêcher la sauvegarde par *dump* d'un fichier ou répertoire situés dans un système de fichiers monté avec l'option dump positionnée à 1 (cf [*fstab*](http://www.lea-linux.org/cached/index/Fstab.html)).

**immutable (i),** « immuable » ─ *réservé à root*

Protège le fichier contre toute modification (suppression, modification, déplacement, changements de permissions), même par root, et empêche qu'on crée un lien vers lui. Seule la modification des attributs étendus est permise à son propriétaire et seul root peut enlever l'attribut « immuable » d'un fichier.

Cet attribut est sans doute l'un des plus intéressants.

**data journalling (j)**, « journalisation des données » ─ *réservé à root*

Cet attribut n'est utile qu'avec une partition en *ext3* montée avec les options de journalisation des données data=ordered (montage par défaut) ou data=writeback pour simuler une journalisation du type data=journal, où les données sont écrites dans le journal avant de l'être dans le fichier lui-même. Consulter man mount 8 pour plus de détails sur les mécanismes de journalisation d'*ext3*.

Sur une partition *ext3* en mode data=journal, cet attribut est ignoré, de même que sur les partitions en *ext2* et *reiserfs*, le premier type n'étant pas journalisé, le second l'étant d'une manière incompatible.

**secure deletion (s)**, « effacement sécurisé » ─ *expérimental*

Un fichier portant cet attribut est non seulement « effacé » mais en plus les blocs qu'il occupait dans le système de fichiers sont recouverts de zéros pour éviter qu'on puisse récupérer les données malgré l'effacement.

Normalement, effacer un fichier consiste seulement à détruire toute référence vers l'espace occupé par les données de ce fichier dans la partition. Il n'est cependant pas question de détruire physiquement les données elles-mêmes, sauf si le fichier possède cet attribut, auquel cas non seulement il n'est plus accessible mais en plus tous ses bits sont réécrits.

**no tail-merging (t)**

**undeletable (u),** « ineffaçable » ─ **expérimental**

Lors d'un effacement, les références vers l'espace du système de fichiers qui contient les données sont détruites, mais pas les données. Sans ces références, cependant, l'espace anciennement réservé peut être utilisé par des fichiers nouvellement créés ou ayant été augmentés. Dans ce cas, il n'est plus possible de récupérer les données en question.

L'attribut « ineffaçable » permet de préserver les blocs qui contenaient les données de façon à ce qu'on puisse les récupérer si besoin est. L'espace-disque n'est donc pas libéré.

**no atime updates (A)**, « pas de mise à jour de la date de dernier accès »

À chaque fois qu'on accède à un fichier, la date du dernier accès est modifiée. Dans un système de fichiers dans lequel existent de nombreux fichiers auxquels on accède souvent (c'est le cas pour les serveurs de *news*), ces modifications peuvent ralentir le système.

Donner à un fichier l'attribut *no atime updates* permet d'éviter ce problème. Noter que la même option existe lorsque l'on monte une partition dans l'arborescence globale (mount -o noatime), option que l'on peut inscrire dans [/etc/fstab](http://www.lea-linux.org/cached/index/Fstab.html).

**synchronous directory updates (D)**, « mise à jour synchone des répertoires »

**synchronous updates (S)**, « mise à jour synchrone »

Ces deux attributs ont un effet similaire. Le premier sert aux répertoires, le second aux fichiers. Les données des répertoires et fichiers concernés sont mises à jour dès qu'une modification a eu lieu (et non en différé, comme c'est le cas par défaut pour *ext2* et *ext3*). C'est l'équivalent des options de montage mount -o sync et mount -o dirsync (qu'on peut aussi passer dans *fstab*) mais pour une sous-partie de l'arborescence et non la partition entière.

**top of directory hierarchy (T)**, « répertoire racine » ─ *réservé à root*

Cet attribut indique au répartiteur de blocs Orlov (ou [*Orlov block allocator*](http://lwn.net/Articles/14633/), utilisé par le noyau Linux pour les systèmes de fichiers *ext2* et *ext3* depuis la version 2.5.46) que le répertoire concerné doit être considéré comme un répertoire racine de la hiérarchie des répertoires.

L'allocateur de blocs est un processus propre au système de fichier permettant de répartir les blocs de données des fichiers dans l'espace disponible tout en garantissant une qu'il ne se produira pas une trop grande fragmentation. Pour cela, on place généralement dans un même groupe de cylindres du disque dur les blocs de fichiers dépendant d'un même répertoire. Dans le cas contraire, les données risquent d'être éparpillées, voire fragmentées (les blocs d'un même fichier sont alors répartis sur plusieurs cylindres non contigus). On ne peut cependant prévoir à l'avance de quel espace aura besoin un répertoire racine. Il pourrait ne contenir que peu de fichiers ou, au contraire, une arborescence complète et chargée. Le répartiteur Orlov prend donc soin d'éloigner le plus possible les uns des autres les répertoires racines, pour qu'ils disposent d'un espace suffisant où croître.

Ajouter cet attribut à un répertoire permet au répartiteur de le traiter comme un répertoire racine, même s'il n'en est pas un, et de le placer loin des autres répertoires racines. Il est donc utile avec des sous-arborescences de fichiers complexes et lourdes qui risqueraient d'être fragmentées.

### Commande lsattr

Cette commande fonctionne comme *ls* et prend les options -R (listage récursif), -a (tout afficher) et -d (lister les répertoires comme des fichiers), entre autres ; *lsattr* peut afficher des attributs qu'on ne peut modifier avec *chattr*

L'affichage est simple : chaque attribut du fichier est représenté par son code, une absence d'attribut l'étant par un -, à la manière du codage des permissions pratiqué par *ls -l* (-rw-r--r--).

Ainsi : chattr +ASacDdijsTtu fichier

lsattr fichier

suS-iadAc----j-tT fichier

L'attribut D n'a pas été appliqué, puisque le fichier n'est pas un répertoire. Les autres valeurs que peut indiquer *lsattr* s'intercalent dans les espaces vacants : *I* (fichier dépendant d'un répertoire indexé par HTree), *E* (fichier dont la compression est erronée ; dépend directement de l'attribut *compressed*), *X* (fichier compressé dont une partie des données brutes peut être consultée directement ; dépend de c) et *Z* (fichier compressé corrompu ; 'c). Ce sont des valeurs non modifiables ajoutées automatiquement aux fichiers. Les trois dernières ne sont toujours pas implémentées.

Noter que *ls* n'indique pas la présence éventuelle d'attributs étendus et que *find* ne peut rechercher des fichiers selon ce critère.

### ls, cp, mv

Les commandes classiques de listage, copie et déplacement ne gèrent pas les attributs étendus, soit qu'elles ne les affichent pas (*ls*), soit qu'elles les ignorent.

### Sauvegarde des données

Sauvegarder des données dotées d'attributs étendus ne peut se faire correctement que si  le système de fichiers de stockage gère lui-même les attributs ;

* le logiciel de sauvegarde est compatible avec les attributs. Dans le second cas, il convient de ne pas utiliser *cpio*, *tar* ou *rsync*, par exemple, mais une application comme *star*.

**4-2 ) LES ACLs**

Les ACLs (Access Control Lists = Listes de contrôle d’Accès) permettent de gérer les permissions caractérisant les autorisations d’accès à un fichier de façon beaucoup plus fine qu’avec les mécanismes traditionnels.

Au moyen des ACL, on peut étendre le nombre d’utilisateurs et de groupes ayant des droits (r,w,x) sur un même fichier. Rappelons que dans le monde UNIX, chaque fichier ne peut normalement indiquer des permissions que sur un seul utilisateur et un seul groupe, qui s’opposent à une unique catégorie correspondant à ‘tous les autres » ( ou « le reste du monde »). Avec les ACLs, on peut ajouter d’autres utilisateurs ou groupes à un fichier et définir leurs droits séparément.

Il existe deux commandes essentielles pour gérer les ACL : **setfacl** et **getfacl**

La commande ***setfacl*** permet d’ajouter ou de supprimer des droits. Les droits peuvent être donnés au niveau d’un utilisateur (u) ou d’un groupe (g).

Vous pouvez visionner les droits ACLs avec la commande **getfacl**.

Forme générale de la commande **setfacl**

**setfacl –m** (pour ajouter ou supprimer des droits) **u :** (si les droits concernent un utilisateur ou g : si c’est un groupe) : nom de l’utilisateur ou du groupe : droits (**r w x**) nom fichier ou répertoire

Exemple :

Je suis sur root et je crée 4 utilisateurs .

Voici leur définitions dans le fichier /etc/passwd

user1 :x :500 :500 ::/home/groupe1/user1:/bin/bash

user2 :x :501 :501 ::/home/groupe2/user2:/bin/bash

user3 :x :502 :502 ::/home/groupe3/user3:/bin/bash

user4 :x :503 :503 ::/home/groupe4/user4:/bin/bash

user5 :x :504 :500 ::/home/groupe1/user5:/bin/bash

user6 :x :505 :501 ::/home/groupe1/user6:/bin/bash

Voici la description des groupes dans le fichier /etc/group

groupe1 :x :500 :

groupe2 :x ;501 :

groupe3 :x :502 :

groupe4 :x :503 :

Je me connecte sous l’utilisateur user1

A la commande **pwd**,

nous avons le résultat **/home/groupe1/user1**

A la commande **ls –lsd /home**,

Nous avons le résultat **8 drwxr-xr-x 7 root root 4896 nov 21 08 :02 /home**

A la commande **ls –lsd /home/groupe1** ,

Nous avons le résultat  **8 drwxr-xr-x 7 root root 4896 nov 21 08 :02 /home/groupe1**

A la commande  **ls –lsd /home/groupe1/user1**

Nous avons le résultat  **8 drwx------ 7 root root 4896 nov 21 08 :02 /home/groupe1/user1**

A la commande  **ls –lsd /home/groupe1/user1/fichier\_acl**

Nous avons le résultat **8 –rwx------ 1 user1 groupe1 62 nov 21 13 :03 /home/groupe1/user1/fichier\_acl**

Je me connecte sous user5 (groupe 1)

A la commande **pwd**,

Nous avons le résultat **/home/groupe1/user5**

A la commande **cd ../user1**

Nous avons le résultat :  **permission non accordée**

Je me connecte sous user2

A la commande **pwd**,

Nous avons le résultat **/home/groupe2/user2**

A la commande **cd ../../groupe1/user1**

Nous avons le résultat :  **permission non accordée**

Je me connecte sous user1.

Je veux que mon fichier **/home/groupe1/user1/fichier\_acl** soit accessible par uniquement par l’utilisateur user2.

Comment procéder ?

Tout d’abord, il faut que user2 puisse traverser le répertoire de user1

Donc je vais lui donner le droit *x*, je suis dans le répertoire  **/home/groupe1**

**setfacl –m u :user2 :x user1**

A la commande  **ls –lsd user1,** j’ai le résultat suivant :

**8 drwx –x---+ 15 user1 groupe1 4896 nov 21 08 :09 user1**

A la commande **getfacl user1**, j’ai le résultat suivant

**# file : user1**

**# owner : user1**

**# group : groupe1**

**user ::rwx**

**user :user2 :--x**

**group ::---**

**mask ::--x**

**other ::---**

Puis ensuite dans le répertoire user1, je tape la commande :

**setfacl –m u :user2 :rx fichier\_acl**

Le résultat de la commande **ls –ls fichier\_acl** est :

**8 –rwxr-x---+ 1 user1 groupe1 62 nov 24 08 :57 fichier\_acl**

A la commande **getfacl fichier\_acl**, j’obtiens le résultat suivant :

**# file : fichier\_acl**

**# owner : user1**

**# group : groupe1**

**user ::rwx**

**user :user2 :r-x**

**group ::---**

**mask ::r-x**

**other ::---**

Seul, l’utilisateur user2 a le droit d’accéder à ce fichier. Tous les autres, y compris user5 qui fait parti du même groupe que user1 a l’accès interdit.

**Attention, les droits visibles par la commande ls –ls ne correspondent plus à ceux du système unix de base.**

Pour enlever tous les acl d’un fichier ou d’un répertoire, il faut simplement faire la commande

**setfacl –b nom-fichier ou répertoire**

exemple :

**setfacl –b fichier\_acl**

|  |
| --- |
|  |

**5 - ) COMMANDES DE GESTION DE FICHIERS ET REPERTOIRES**.

**5-1) manipulations des répertoires**.

* **pwd** (**print working directory**) : affiche le répertoire courant.
* **cd [répertoire] (change directory)**

Exemple : **cd**

La commande sans arguments, rend le répertoire personnel, spécifié dans la variable **HOME**, le répertoire courant.

**cd /home/users**: on précise le chemin à partir de "**/"**

**cd ..** remonte au niveau du répertoire père

* **ls [options] réf** lister le contenu d'un répertoire (par défaut, le répertoire courant).

Quelques options :

**a** : liste toutes les entrées d'un répertoire y compris les noms commençant par "**.**"

**i** : pour lister les numéro d'**i-node** des fichiers et répertoires.

**l** : format long donnant:

le type de fichier

le mode du fichier (droits d'accès)

le nombre de liens physiques

le propriétaire

le nom du groupe

la taille (nombre de caractères). Si le fichier est un fichier spécial, à la place de la taille, on obtient les informations correspondant à l'identification de la ressource correspondante.

la date de dernière modification

il est possible de combiner les options.

Exemple : **ls -ls**

* **mkdir [-option] réf …**

création de répertoire vide (ne contenant que "**.**" et "**..**" respectivement le répertoire courant et le répertoire père)

* **rmdir [-f] [-i] [-p] réf\_répertoire**

permet de supprimer une entrée correspondant à un (des) répertoire(s) vide(s), ne contenant que les entrées "**.**" et "**..**"

**f**  : la suppression est forcée (annulation de l'effet "**i**"); elle supprime également les messages provoqués par la non existence de paramètres.

**i** : chaque suppression est précédée d'une demande de confirmation; cette option est ignorée si "**f**" est utilisée

**p** : provoque le suppression du répertoire père si celui-ci est vide après la suppression d'un répertoire. Le processus de suppression se poursuit jusqu'au premier répertoire non vide. Si l'option "**i"** est également utilisée, chaque suppression est précédée d'une demande de confirmation.

Il est impossible pour un processus de supprimer son répertoire de travail. Cependant, un processus peut supprimer le répertoire de travail d'un autre!

**5-2) Commandes sur les fichiers.**

* **cat [-options] [ref …]**

lit les fichiers référencés et écrit leurs contenus sur la sortie standard. **<CTRL-S>** pour arrêter le défilement, **<CTRL-Q>** pour reprendre le défilement.

- **more réf\_fichier**

affichage page par page sur un écran du contenu d'un fichier

* **cp [-otions] réf\_source [réf\_source2] réf\_dest**

réf\_dest : fichier ou répertoire

recopie physique d'un ou plusieurs fichiers de références données.

Options:

**i** : demande de confirmation pour chaque fichier existant

**r** : si l'une des ref\_sources est celle d'un répertoire, la commande recopie la sous-arborescence ayant cette référence comme racine

* **mv [-f] réf\_fichier réf**

**mv [-f] réf-1 [réf\_2…] réf\_répertoire**

**mv [-f] réf\_rep1 [réf\_rep2…] réf\_repertoire**

permet de renommer ou de déplacer une ou un ensemble de référence de fichiers

**f** : suppression du message de confirmation

* **rm [-f] [-i] réf\_fichier**

**rm –r [-f] [-i] réf-répertoire … réf\_fichier**

efface le ou les fichiers spécifiés en paramètre

**f** : suppression du message de confirmation

**i** : demande de confirmation

**r** : permet de supprimer un répertoire, son contenu est supprimé ainsi que d'éventuel sous-répertoires et leurs contenus, récursivement.

* **ln [-s] fichier\_ref fichier\_dest**

**ln –s ref\_répertoire répertoire\_dest**

Un même fichier (caractérisé par son **inode**) peut avoir plusieurs noms (dans la même répertoire ou non)

Un lien hard ne peut être fait que sur un fichier situe sur un même file system (même **inode**)

Si les fichiers sont sur des "**file system"** différents, alors on ne peut faire que des liens symboliques.

Un lien sur un répertoire ne peut être qu'un lien symbolique qu'il soit ou non situé sur un même file system

Remarques importantes :

**Bien faire la différence entre ln et cp** (cp duplique physiquement le fichier)

**rm** détruit le lien (décrémente de 1 le nombre de lien: le fichier n'est donc vraiment détruit que lorsque le nombre de lien est de 1)

Avec un lien hard-link, plus rien ne permet de distinguer le fichier original du nouveau fichier crée par **ln**

Il existe aussi des liens symboliques qui s'obtiennent par ma commande **ln –s**. Le lien symbolique est simplement un pointeur vers un autre nom. Attention, si on détruit le fichier ou le répertoire original sans détruire le fichier ou répertoire lien, il y aura une incohérence entre **la table des** **inodes et le système de fichier**s.

**5-3 ) Comparer le contenu des fichiers**.

**- cmp [-option] ref\_fichier1 ref\_fichier2**

Comparaison de fichiers . S'ils sont différents, le numéro de la première ligne contenant une différence est affichée ainsi que la position (dans le fichier) du caractère différent (exemple : char1, ligne1)

Options:

**l** : toutes les différences sont signalées

**s** : aucune impression de différences n'est réalisée; renvoie d'un code de retour (0 : fichier identique, 1 : fichiers différents, 2 : en cas d'erreur (fichier inexistant par exemple)

- **diff [-option] réf\_fichier1 ref\_fichier2**

compare les contenus de deux fichiers.

**5-4) Compter le nombre d'élément d'un fichier**

- **wc [-lwc] [réf\_fichier ]**

compte pour chaque fichier (ou l'entrée standard s'il n'y a pas de paramètres) le nombre de lignes(l), de mot(w), ou de caractère(c) qu'il contient. Par défaut, les trois valeurs sont fournies.

**5-5) Réunir les informations de plusieurs fichiers**.

- **paste [-s] réf\_fic1 réf\_fic2**

permet de joindre les lignes de deux fichiers, les unes après les autres. Les lignes sont séparées par le caractère de tabulation. L'option **–s** permet de mettre les différentes lignes d'un fichier en série l'une derrière l'autre.

**5-6) Classification des fichiers**.

- **file [-option] réf …**

par une série de test sur le contenu du fichier ou les informations de **l'i-node** de ce fichier, la commande essaie de classifier le fichier (ASCII, binaire répertoire, source C, …)

**5-7) Trier un fichier**

- **sort [-options] [+pos1] [-pos2] [réf\_fic …] [-o réf\_fic\_sortie]**

trie les différentes lignes du (ou des) fichier(s) et écrit les résultats sur la sortie standard, sauf si l'option "**o"** est utilisée pour construire un fichier de sortie. Le tri est un tri lexicographique sur l'ensemble des fichiers d'entrée.

Options :

**U** : un seul exemplaire des lignes identiques est conservé

**R** : l'ordre est inversé

**F** : les lettres majuscules et minuscules correspondantes sont considérées comme identiques

…

Par ailleurs, lorsque la notation [[+pos1] [-pos2]] est utilisée, les lignes sont découpées en champs (un champ= une suite de caractères suivie d'un séparateur ou d'un caractère de fin de ligne). Dans cette notation pos1 et pos2 sont de la forme <n1>[<n2>]. Elle permet de définir un ensemble de positions sur lesquelles porte la comparaison.

Exemple : jmm enseignant bureau 29

La commande "sort +2.6" prendra les deux derniers caractères 29 comme clé de tri. La commande "sort +2.6" signifie après les deux derniers caractères 2 premiers champs et après les "6 premiers caractères".

**5-8) Sélectionner des données dans un fichier**.

**- split [-nombre] [ref\_fichier]**

lit le fichier de référence donné (l'entrée standard par défaut) et crée des fichiers obtenus en le découpant en morceaux de nombre de lignes (par défaut 1000). Les noms des fichiers obtenus ont comme préfixe la chaîne donnée en paramètre et comme suffixes successifs aa, ab, ..,zy,zz

- **head [-nombre] [réf\_fichier …]**

affiche les premières lignes d'un fichier. Le nombre de lignes étant passé en paramètre (-nombre). Par défaut, la commande affiche les dix premières lignes.

- **tail [-nombre] [réf\_fichier]**

affiche les dernières lignes d'un fichier. Le nombre de lignes étant passé en paramètre (-nombre). Par défaut, la commande affiche les dix dernières lignes

**5-9 ) Retrouver un fichier ou répertoire dans l'arborescence**

La commande **find** permet de rechercher des fichiers ou des répertoires à l'intérieur de l'arborescence.

Syntaxe :

**find**  "nœud de l'arborescence où débute la recherche" options

Où options précise les critères de sélection.

Les principales options sont :

-**name** fich recherche des fichiers de nom fich

-**type** x recherche des fichiers ayant un type x (ex: f fichier, d répertoire)

-**user** nom fichier appartenant à l'utilisateur nom

-**group** nom fichier appartenant au groupe nom

-**size** n fichier de taille n (en bloc)

-**atime** n fichier ayant eu un accès en n jours

-**mtime** n fichier modifié n jour auparavant

-**exec** cmd permet l'exécution d'une commande cmd sur les fichiers qui satisfont les autres options.

-**print** permet l'affichage du résultat

Ces options peuvent être combinées (cela équivaut à un ET des options).De plus, on peut utiliser des parenthèses, et les opérateurs **!** (négation) et **–o** (OU)

Exemples :

Recherche des fichiers se nommant core et les effacer

**find / -name core –exec rm {} \;**

Recherche des répertoires s'appelant tmp

**find / -name tmp –type d –exec ls –ls {} \;**

**5-10 ) Rechercher des mots dans un fichier**.

**- grep [-option] expression\_régulière [réf\_ordinaire …]**

**commande grep, egrep fgrep**

Sélection dans un fichier des lignes contenant un motif satisfaisant une expression régulière donnée.

Options :

**c** : seul le nombre de ligne satisfaisant l'expression est écrit

**i** : pas de distinction majuscule / minuscule

**l** : seuls les noms de fichier satisfaisant la recherche sont écrits

**n** : chaque ligne est précédée de son numéro dans le fichier qui la contient,

**codes retour possible (pas de référence)**

**0** : au moins une ligne satisfaisante

**1** : aucune ligne satisfaisante

**2** : erreur de syntaxe ou (au moins un) fichier inaccessible.

**Les expressions régulières**.

Les règles des expressions régulières sont les suivantes :

**grep str** : recherche le mot "str"

**grep s.r** : recherche tous les mots de 3 caractères commençant par "s" et se terminant par "r".

Le "." remplace n'importe quel caractère.

**grep [sS]tr** : recherche de "str" ou "Str". Il est possible de spécifier un ensemble de caractères en indiquant "a-z".

**[^sS]tr** : recherche de mots ne commençant par "s" ou "S" et se termine par tr.

**.** : représente n'importe quel caractère ( attention 1 seul caractère!)

**\*** : représente n'importe quelle séquence du caractère précédent. Lorsque le caractère "\*" suit le caractère ".", cela veut dire n'importe quelle chaîne de caractères.

**^ ou $** : recherche du mot en début de ligne(^) ou en fin de ligne ($).

**\** : permet d'ignorer l'effet du caractère spécial suivant.

**+** : au moins une occurrence de l'expression régulière.

**|** : joue le rôle de "ou" entre 2 expressions régulières.

**\{a,b}\** : permet d'indiquer un nombre d'occurrences du caractère précédent.

"a\{3\}" recherche la séquence "aaa"

"a\{3,\}" recherche la séquence d'au moins 3 caractères "a"

"a\{3,4\}" recherche les séquence "aaa" et "aaaa"

**\{mot\}** : permet de donner un numéro d'ordre à un mot.

Exemple : \(ping\)\(pong\)\2\1, effectue la recherche de "ping pong ping pong"

Ecriture des motifs :

- un caractère correspond à lui-même, en règle générale.

Ainsi le motif c recherche le caractère "c" désigné.

- un méta caractère précédé de "\", pour lui rendre son rôle usuel.

-" ." le point remplace tout caractère unique sauf \n (newline)

"a." toute suite de 2 caractères commençant par a sauf a\n

"b.c" désigne toute suite de 3 caractères du genre bac, bbc, bcc …

- [abc] classe de caractère, sélectionne toute chaîne contenant l'un des caractères listés :

[ a-z] toute lettre minuscule

[0-9] équivaut à [0123456789], un chiffre quelconque.

[a-zA-Z0-9\-\_] correspond à n'importe quelle lettre ou chiffre, ou au tiret ou au souligné.

- le caractère "^" juste après " [" joue le rôle d'exclusion des caractères qui suivent.

[^aeiou] tout sauf une voyelle

[^a-zA-Z0-9] sélectionne un caractère non alphanumérique

Remarques:

Les symboles ^, $ sont appelés motifs d'ancrage, car ils permettent de préciser l'endroit où doit être effectué la recherche du motif dans la chaîne (alors que sans directive particulière, la recherche de correspondance du motif s'effectue de gauche à droite de la chaîne).

Le symbole d'ancrage "\b" indique une limite de mot obligatoire, à l'endroit où il est inséré dans le motif

Exemple : info\b est vérifié par 'l'info pour tous", "info-matin" mais pas par infos

b\info est vérifié par "informatique, "m'informer" mais pas désinformation, …

# GESTION DES DISQUES

Un système de gestion de fichiers (file system) est l'entité regroupant les fichiers mémorisés sur disque. il contient :

* les données des fichiers
* un ensemble d'informations techniques (utilisées par Unix pour assurer la gestion du système de fichiers)

Les accès physiques se font pour chaque type de disque par une carte interface qui connecte le disque à l'architecture de bus de l'ordinateur. Le système d'exploitation accède logiquement au disque physique à la fois par le gestionnaire de périphérique et par les fichiers spéciaux de périphériques. On peut visualiser les fichiers spéciaux en listant le répertoire **/dev/dsk**.

## Structure physique d'un système de fichiers

Chaque système de fichiers est constitué de:

* **Bloc d'initialisation** (boostrap) utilisé au chargement du système. Tout SGF en possède un, même s'il n'est pas utilisé au chargement du système.
* **le superbloc** contient les informations générales sur le SGF (taille du système de fichiers, nombre de blocs libres, le début de la liste des blocs libres, un pointeur sur le premier bloc libre de la liste, la taille de la liste des i-nodes, le nombre et la liste des i-nodes libres, etc…)
* **la table des i-nodes** : chaque i-node contient
  + le type et les droits d'accès au nœud pour les différents utilisateurs
  + l'identité du propriétaire et du groupe propriétaire
  + le nombre de liens physiques
  + la taille du fichier
  + les dates de création, de dernière consultation, de dernière modification
  + l'adresse de blocs de données du fichier

La commande qui permet de déclarer un disque est la commande **mknod**

Syntaxe : **mknod name b/c majeur mineur**

où

- **name** nom du périphérique

- **b/c** b déclaration en mode bloc, c en mode caractère

- **majeur** numéro référençant un driver

- **mineur** numéro identifiant le périphérique

Pour les numéros de majeur et mineur, il convient de se reporter à la documentation du constructeur.

**Découpage physique du disque**.

Le découpage du disque, c'est le partionnement suivant le nombre de blocs physiques.

**Disque virtuel**.

Un **disque virtuel** ou **partition**, est un sous-ensemble d'un disque physique. La déclaration d'un disque virtuel se fait de la même manière qu'un disque physique, par la commande **mknod**.

Après avoir découpé physiquement le disque, et créé les disques virtuels par **mknod**, ces espaces peuvent être structurés pour en faire des systèmes de fichiers (**file system**).

La commande **mkfs** permet de créer ces systèmes de fichiers :

Syntaxe :

**mkfs name size**

où

- **name** nom du disque virtuel

- **size** taille de celui-ci

**Action de mkfs**

* création du super-bloc
* création de la table des i-nodes
* création de la liste des blocs libres

Un répertoire **lost+found** doit être créé sur la racine de chaque file system. Il est nécessaire, en particulier, pour la commande **fsck. Le répertoire lost+found est créé par la commande mklost+found**

**La commande fsck**.

La commande **fsck** est le principal outil de maintenance du système de fichiers pour contrôler la cohérence du système de fichiers et effectuer les réparations éventuelles. N’exécutez jamais **fsck** sur un système monté.

Cependant, **fsck** doit être exécutée régulièrement pour assurer l’intégrité structurelle du système de fichiers :

* **fsck** est appelée à l’amorçage du système par un script appelé par **init**.
* Pour une maintenance préventive , **fsck** doit être exécutée de temps en temps sur tous les systèmes de fichiers.

Pour effectuer ces contrôles, **fsck** examine plusieurs fois le système de fichiers, examinant à chaque fois différentes caractéristiques, dont :

* la taille de bloc et de fichier
* les noms des chemins
* la connectivité (relation père fils)
* les liens et nombre de références
* les groupes de cylindres

La commande ne doit pas être exécutée sur le système de fichiers racine qu’à partir du niveau d’exécution "**s**" d’**init**, le niveau d'exécution de l’administrateur système. Appelée sans options, **fsck** s'exécute inter-activement sur les systèmes de fichiers locaux déclarés dans **/etc/fstab** et vous demande une réponse lorsqu’il décèle une incohérence.

**Montage de système de fichiers.**

Pour être exploités, les systèmes de fichiers doivent être montés logiquement sur un répertoire de l’arborescence du système de fichiers. Le choix du répertoire incombe à l’administrateur, celui-ci peut prendre un répertoire existant ou le créer par la commande **mkdir.**

Le répertoire sur lequel on monte un système de fichiers est directement situé sous la racine du système ( **/** ).

L’ordre de montage des files system est la commande **/sbin/mount** permet de rattacher le système de fichier à l’arborescence existante.

Le montage des files system s’effectue à chaque lancement du système. La commande **/sbin/umount** effe**c**tue l’opération inverse.

Certains fichiers d’administration doivent être mis-à -jour après une configuration disque :

* **/etc/mnttab** contient la liste des fichiers devant être montés à chaque lancement.
* /**etc/fstab** contient la liste des files system à vérifier sous **fsck**

### Le fichier /etc/fstab

Le nom **fstab** signifie : File System Table

#### Syntaxe du fichier /etc/fstab

Exemple :

# ceci est un commentaire dans **/etc/fstab**

On peut mettre dans **/etc/fstab**, autant de lignes vides que l'on veut (elles sont bien évidemment ignorées). Enfin les seules lignes ayant une signification ont une syntaxe relativement figée : chaque ligne doit contenir 6 champs (au maximum, 4 au minimum), chaque champ étant séparé des autres par un espace (un espace ou une tabulation)

Les 6 champs sont :

**1 – périphérique** :

Le premier champ de chaque ligne de **fstab** doit contenir, pour être valide un nom de fichier, en principe, un périphérique, c'est-à-dire un fichier de **/dev**/

exemple : /dev/hda5 ( 5ème partition du 1 er disque IDE)

/dev/sda5 ( 5ème partition du 1 er disque SCSI)

La première colonne peut avoir une valeur "**none**". Il correspond, par exemple au répertoire **/proc** qui est utilisé de façon interne par le **kernel** : il ne contient réellement des fichiers, ceux qui peuvent y apparaître ne sont que des images d'informations contenues dans le **kernel**.

**2 – point de montage** :

Le second champ sert à indiquer à la commande **mount** le répertoire (point de montage) qui sera associé à ce file système. C'est le répertoire sur lequel on va associer la partition. Remarquez la ligne contenant le mot **swap** : elle désigne en fait la mémoire virtuelle, elle ne se monte pas.

**3- système de fichier** :

Ce champ indique à la commande **mount** quel est le type du système de fichiers contenu sur le périphérique.

"**ext2**" est le système Ext2Fs utilisé normalement par Linux.

"**vfat**" est le système de fichier de Windows 95/98

"**iso 9660**" est utilisé par les CD-ROMs

En ce qui concerne **/dev/fd0**, qui correspond à la disquette, nous avons "**auto**" comme type de fichier. Une disquette peut être formatées avec différents systèmes, "**auto**" demande à linux de reconnaître automatiquement le type de formatage de la disquette.

**4- options**

Ce champ est un peu particulier, car il indique à la commande **mount** les paramètres de montage du file system.

Le mot "defaults" indique qu'il faut appliquer les options par défaut, ce qui inclus entre autre le montage automatique : toutes les partitions ainsi marquées, seront montées avec la commande "**mount –a**"

Il existe beaucoup d'options de montage en particulier :

**nosuid** : cela indique que le bit "s" sera inhibé (important pour les files system

**noauto** : ces périphérique ne seront pas montés automatiquement

**ro** : montage en lecture seule (read only), en général pour les CD-ROM

**5- sauvegarde**

S'il est non nul, ce champ indiquera à l'utilitaire **dump** qu'il peut sauvegarder ce système de fichiers. Mettez "**1**" pour une partition "**ext2**" et "**0"** pour les autres.

**6- réparation/test**

Ce champ indique à la commande **fsck** qu'il faut tester la partition. Lorsque ce champ est égal à "**1**" cela signifie que la partition doit être testée en premier, "**2**" pour les autres partitions "**ext2**".

Normalement le file system abritant le répertoire racine, doit être monté en premier, d'où la valeur "**1**" .

Les autres peuvent être vérifiés dans n'importe quel ordre, mais *après* la racine : ils sont tous à "**2**".

Lorsque la valeur est "**0**" cela signifie que ces files system n'ont pas besoin d'être vérifiés. Cela concerne les disquettes, les CD-ROM mais aussi la **zone swap** ainsi que le pseudo-système de fichiers **/proc.**

### Quelles sont les partitions à créer ?

**a) la partition racine : la partition /**

Une partition petite (environ 200 Mo) pour la partition **/**. Cette partition recevra **/etc**, **/bin**, **/sbin** …. Ces répertoires ne bougeront pas en taille tout au long de la vie du système.

Le montage de la partition racine est très important : sans lui, vous avez toutes les chances d'avoir un "*kernel panic*" : c'est-à-dire que lors du **boot**, le **kernel** ne trouve pas la partition racine, et se bloque sur ce message.

Le premier champ de ce type de montage doit être le nom du périphérique étant réellement la racine de votre installation. En fait, vous n'avez que très peu de raison de vouloir changer cette ligne. L'option "***defaults***" n'est par contre, pas forcément la plus adaptée. Vous pouvez parfois vouloir limiter les accès à cette partition ou autre. Le sixième champ doit être égal à "**1**" pour que **fsck** commence par la partition "**/**"en cas de problème.

**b) la partition swap**.

Une partition qui possède "**swap**" comme type de file système, signifie que cette partition servira de mémoire virtuelle, c'est-à-dire de zone d'échange entre la mémoire et le disque.

**c) les autres partitions**.

Un administrateur a le choix de découper son système de fichiers sur plusieurs "file system". Il n'y a pas d'obligation à suivre.

Quelques conseils :

**partiton /usr**

Une partition /usr (environ 2,5 Go) : Cette partition doit être importante parce qu'elle va recevoir les programmes non indispensables au système de base.

Il y aura aussi /usr/local qui sont des programmes développés par l'administrateur ou des produits installés en dehors du système de base.

**partition /var**

Cette partition contiendra, en particulier, le courrier, le spool, les fichiers traces appelés **logs**. Sa taille va variées dans le temps. Il faut créer une partition **/var** parce qu'en cas de problème système ou d'un produit, les messages d'erreur dirigés dans les fichiers **logs** ne satureront pas la partition "**/**"

**partition /tmp**

Cette partition sert pour l'utilisation des fichiers temporaires.

Petit conseil : il faut faire un lien symbolique entre tous les répertoires temporaires (**/var/tmp**, **/usr/tmp**, **/var/preserve** …)

**partition /home**

Cette partition recevra tous les répertoires personnels des utilisateurs. sa taille dépendra du nombre d'utilisateur déclarés dans le système.

### Les partitions /proc et /dev/pts

Ce ne sont pas réellement des partitions, leur périphérique correspondant à ces partitions est "none".

Ces partitions sont des pseudo-systèmes de fichiers situés en mémoire vive, qui représentent des informations sur la machine sous la forme de simples fichiers, manipulables avec les commandes standard (**cat**, **more** etc…)

**Commande df**

La commande **df** permet de lister l'ensemble des files system montés le système.

Avec des options, nous pouvons connaître le pourcentage utilisé ainsi que le pourcentage libre

##### Commande du

La commande **du** permet de connaître la place que prend un fichier, un répertoire et sous-répertoire(s).

**EDITEUR DE TEXTE**

**Utilisation de l'éditeur vi**

Un éditeur de texte est un programme destiné à créer ou modifier aisément des fichiers qui contiennent du texte, comme un courrier, un programme en langage C ou un fichier de configuration.

**Bien qu'il y ait beaucoup de tels éditeurs disponibles sous Linux, le seul que vous soyez assuré de trouver sur tous les systèmes UNIX du monde est vi.**

Le "Visual editor".vi est loin d'être le plus facile à utiliser, ou le plus convivial. Toutefois il est si répandu dans le monde UNIX, et il peut tellement vous être indispensable par moments, que nous devons en parler un peu ici.

Le programme vi est très puissant, mais un peu compliqué à utiliser. Malgré tout, une fois que vous aurez pris le temps (en vous énervant quelquefois...) d'apprendre à vous en servir, vous finirez par trouver vi facile et souple.

Comme beaucoup de choses, c'est souvent l’apprentissage qui est pénible.

# Concepts

Sous vi, vous êtes à tout moment dans l'un des trois modes d'opération possibles. ces modes sont

* le mode commande
* le mode édition,
* le mode dernière ligne.

**Un travail effectué sous vi est toujours fait en mémoire tampon. Son enregistrement doit être demandé pour que votre travail soit enregistrer sur le disque dur.**

# Mode de travail de vi

## Enchaînement des modes de travail en vi

**K**

**h**

**I**

**j**

Lorsque vous lancez **vi**, vous êtes en mode *commande*. Ce mode vous autorise un certain nombre de commandes permettant d'éditer des fichiers ou de basculer dans un autre mode. Par exemple, taper "**x**" en mode commande efface le caractère qui se trouve sous le curseur. Les touches fléchées déplacent le curseur dans le texte que vous éditez.

Généralement, les commandes utilisées dans ce mode font un ou deux caractères de long.

Pour insérer ou écrire du texte vous devez utiliser le mode édition. C'est probablement celui sous lequel vous passerez le plus de temps lorsque vous utiliserez **vi**. Pour entrer en mode édition, vous devez taper (depuis le mode commande) " **i** ", (pour "*insertion*"), et là vous pouvez taper du texte qui s'insérera à la position courante du curseur.

**Pour sortir du mode édition et revenir en mode commande, pressez simplement la touche esc**.

Le mode **ex** ou mode dernière ligne est un mode spécial utilisé pour passer certaines commandes étendues à vi. Lorsque l'on tape ces commandes, elles apparaissent sur la dernière ligne de l'écran (d'où le nom).

**Pour sortir du mode commande et aller au mode dernière ligne " ex " vous tapez " : "**

Depuis le mode commande, vous passez dans ce mode dernière ligne, et vous pouvez alors taper des commandes comme " **wq** "(qui écrit le fichier et termine le programme) ou " **q!** " (pour quitter vi sans sauver les modifications). Le mode dernière ligne est en général utilisé pour des commandes plus longues qu'un seul caractère.

## Appel de vi

Le mieux pour que vous puissiez assimiler ces concepts est d'appeler vi et d'éditer un fichier tout en lisant ces lignes.

Appelez vi en tapant

**vi toto**

Si toto existe, il l’ouvre sinon il le crée.

La colonne de caractères " ~ " indique que vous êtes à la fin du fichier.

## Insertion de texte

Vous êtes donc en mode commande ; pour insérer du texte dans le fichier, tapez **i** (ce qui vous placera en mode édition), et commencez à taper votre texte.

Vous pouvez entrer autant de lignes que vous voulez lors de l'insertion de texte, (en pressant la touche Entrée à la fin de chaque ligne, bien entendu), et vous pouvez corriger vos erreurs immédiatement par la touche "**backspace**" (retour arrière, celle qui est au-dessus de Entrée en principe).

Pour sortir du mode édition et revenir au mode commande, tapez sur la touche **esc**.

Une fois en mode commande, vous pouvez utiliser les touches fléchées pour vous déplacer dans le texte. Ici, comme nous n'avons encore qu'une ligne, les touches haut et bas seront inopérantes et vous serez alerté par un signal sonore.

Il y a plusieurs façons d'insérer du texte, autres que la commande **i**. Parexemple la commande **a** permet l'insertion de texte **juste après la position du curseur**, et non pas juste dessus.

Pour insérer du texte une ligne au-dessous, utilisez la commande **o**.

Gardez à l'esprit simplement, qu'à tout moment vous êtes soit en mode commande (où les actions comme **i**, **a** ou **o** sont valides) ou en mode insertion (où le texte saisi s'affiche, suivi de **esc** pour revenir en mode commande) ou enfin en mode dernière ligne.

## Suppression de texte

Depuis le mode commande, la touche **x** efface le caractère situé sous le curseur.

Pour supprimer tout le mot sous lequel se trouve le curseur, utilisez la commande **dw**. Si vous aviez tapé les caractères accentués dans le texte, il se peut que la commande ne fonctionne pas sur les mots accentués, selon l'origine de votre version de vi. Bienvenue dans le monde où ceux qui ne parlent pas anglais sont très souvent ignorés.

## Remplacement de texte

La commande **r** est très semblable aux commandes **i** et **a**, mais elle remplace le texte existant au lieu d'effectuer une insertion.

La commande **r** remplace un unique caractère se trouvant sous le curseur.

La commande " **~** " bascule la lettre située sous le curseur de minuscule en majuscule, et vice versa ; Vous constaterez probablement encore une fois que les caractères accentués ne sont pas pris en compte.

## Commandes de déplacement

Vous savez déjà comment utiliser les touches fléchées pour vous déplacer dans le document. Il est aussi possible de se déplacer de la même façon par les commandes **h**, **j**, **k**, et **l**, qui déplacent le curseur respectivement à gauche, en bas, en haut, et à droite. Il est très utile de connaître ces commandes lorsque les touches fléchées ne fonctionnent pas pour une raison quelconque. (Par exemple, si le clavier n'en est pas équipé). Cela permet de se sortir des pires situations.

Mode commande

Mode saisie

Mode ex

vi fichier

zz

:q!

a

i

Echap

:

Entrée

La commande **w** déplace le curseur au début du mot suivant ; la commande **b** le place au début du mot précédent.

La commande **0** (c'est un zéro) déplace le curseur au début de la ligne courante, et **$** le place à la fin de la ligne.

Lors de l'édition de gros fichiers, vous aurez besoin de vous déplacer par écrans entiers pour aller plus vite. Pressez **ctrl-F** pour avancer d'un écran, et **ctrl-B** pour remonter dans le fichier.

Pour aller directement à la fin du fichier, tapez **G**. Vous pouvez également vous positionner sur une ligne arbitraire; par exemple si vous tapez la commande **10G** vous vous retrouverez sur la ligne 10 du fichier. Pour vous positionner au début du fichier, utilisez la commande **1G**.

Vous pouvez coupler les commandes de déplacement avec d'autres, comme la suppression. Par exemple, **d$** supprimera tout le texte à partir du curseur jusqu'à la fin de la ligne, **dG** supprimera tout ce qui se trouve à partir du curseur jusqu’à la fin du fichier, et ainsi de suite.

## Sauver le fichier et quitter vi

Pour quitter vi sans conserver les modifications dans le fichier édité, utilisez la commande **: q!**. Lorsque vous tapez " **:** ", le curseur se déplacera sur la dernière ligne de l'écran et vous serez en mode dernière ligne.

En mode dernière ligne, certaines commandes étendues sont disponibles. L'une d'elle est **q!** ,qui sort de l'éditeur sans sauver le fichier. La commande **:wq** enregistre le fichier puis termine vi. La commande **ZZ** (depuis le mode commande, donc sans le " **:** ") est équivalente à **:wq**.

Pour enregistrer le fichier, mais sans quitter l'éditeur, tapez simplement **:w**.

## Editer un autre fichier

Pour éditer un autre fichier, utilisez la commande **:e**. Par exemple, pour cesser d'éditer toto, et éditer le fichier bidule à la place, utilisez la commande **e bidule**.

Si vous utilisez **:e** sans sauver préalablement le premier fichier, vous aurez le message d'erreur qui signifie simplement que vi ne veut pas changer de fichier tant que vous n'avez pas sauvé le premier, par sécurité. À ce moment vous pouvez soit utiliser **:w** pour enregistrer le fichier en question, puis utiliser **:e**, soit taper la commande.

Le " **!** " indique à vi que vous insistez vraiment pour éditer le fichier et que vous êtes bien au courant que vous allez perdre le premier.

## Inclure un autre fichier

À l'aide de la commande **:r**, vous pouvez inclure le contenu d'un autre fichier dans celui que vous êtes en train d'éditer. Par exemple la commande **:r toto** insère le contenu de toto à la suite de la ligne courante

## 2.10 Lancer une commande shell

Vous pouvez également lancer des commandes depuis l'éditeur vi. La commande **:r!** fonctionne comme **:r**, mais au lieu de lire un fichier, elle insère la sortie de la commande indiquée, à partir de la position courante du curseur.

Lorsque vous sortirez de ce shell (en utilisant sa commande **exit**), vous retrouverez vi.

**:r! Date** insère le résultat de la commande date à la suite de la ligne courante.

## Obtenir de l'aide

Vi n'offre pas beaucoup d'aide interactive (peu de programmes UNIX le font), mais vous pouvez toujours lire la page de manuel qui lui est consacrée (**man vi**). En fait, cet éditeur est une interface plein écran de l'éditeur **ex**; c'est en réalité **ex** qui gère l'essentiel des commandes passées en mode dernière ligne. Aussi, en plus de la lecture de la page de manuel de vi, jetez un œil également à celle de **ex**.

# Tableaux des principales commandes de vi

## Commandes d’activation du mode saisie

|  |  |
| --- | --- |
| **Commande** | **fonction** |
| a | Ajout derrière le caractère courant |
| A | Ajout à la fin de la ligne |
| i | Insertion devant le caractère courant |
| I | Insertion en début de ligne |
| o | Insertion d’une nouvelle ligne, sous la ligne courante |
| O | Insertion d’une nouvelle ligne au-dessus de la ligne courante |

## Caractères de simulation des touches de direction

|  |  |
| --- | --- |
| **Touche** | **Déplacement** |
| h | Un caractère vers la gauche |
| l | Un caractère vers la droite |
| k | Une ligne vers le haut |
| j | Une ligne vers le bas |

## Commandes de positionnement

|  |  |
| --- | --- |
| **Touche** | **Déplacement** |
| 0 (zéro) | Début de la ligne courante |
| $ | Fin de la ligne courante |
| w | Mot suivant |
| b | Mot précédent |
| f+c | Sauter sur le caractère c. Valable uniquement sur la ligne courante |
| ctrl + F | Remonter d’un écran |
| ctrl + B | Descendre d’un écran |
| G | Dernière ligne du fichier |
| nG | Sauter à la ligne n. Exemple 1G pour la 1er ligne ou 8G pour la 8éme |

## Commandes de modification

|  |  |
| --- | --- |
| **Touche** | **Effet** |
| x | Suppression du caractère sous le curseur |
| X | Suppression du caractère devant le curseur |
| r+c | Remplace le caractère sous le curseur par le caractère c |

## Commandes de suppression et de copie

|  |  |
| --- | --- |
| **Touche** | **Effet** |
| dw | Supprime le mot courant |
| d$ ou D | Supprime tous les caractères jusqu’à la fin de la ligne |
| dO | Supprime tous les caractères jusqu’au début de la ligne |
| dfc | Supprime tous les caractères de la ligne courante, jusqu’à la 1er occurrence du caractère c. |
| dG | Supprime tous les caractères jusqu’à la dernière ligne,y compris la ligne courante. |
| DIG | Supprime tous les caractères jusqu’à la 1er ligne, y compris la ligne courante. |
| dd | Supprime la ligne courante |
| u | Annule la dernière suppression |
| yw | Copie le mot courant |
| y$ ou Y | Copie tous les caractères jusqu’à la fin de la ligne |
| yO | Copie tous les caractères jusqu’au début de la ligne |
| yfc | Copie tous les caractères de la ligne courante, jusqu’à la 1er occurrence du caractère c. |
| yG | Copie tous les caractères jusqu’à la dernière ligne, y compris la ligne courante. |
| YIG | Copie tous les caractères jusqu’à la 1er ligne, compris la ligne courante. |
| yy | Copie la ligne courante |

## Commandes couper(déplacement) copier coller

placez vous au début de la sélection ex déplacement de 10ligne

**10dd**

placez vous au début de la sélection ex copie de 10ligne

**10yy**

placez vous ou vous voulez coller

**p** (copie derrière le caractère courant)

**P** (copie devant le caractère courant)

## Commandes de recherche

|  |  |
| --- | --- |
| **Commande** | **effet** |
| / | recherche vers le bas |
| ? | recherche vers le haut |

## Caractères spéciaux de recherche

|  |  |
| --- | --- |
| **Commande** | **fonction** |
| /[aA]n | Recherche tous les mots ayant un A ou un a en 1er lettre |
| /[a-e]n | Recherche tous les mots compris entre a à e comme 1er lettre |
| /[a-eA-E]n | Recherche tous les mots compris entre a à e et A et E comme 1er lettre |
| /[^a-e]n | Recherche tous les mots non compris entre a à e comme 1er lettre |
| /.n | Recherche tous les mots ayant un caractère quelconque comme 1er lettre |
| /a\*n | Recherche tous les mots ayant un a comme 1er lettre et ayant le n répété x fois |
| /^toto | Recherche toto en début d’une ligne |
| /toto$ | Recherche toto en fin d’une ligne |
| /toto\$ | Recherche toto$ (\ verrouillé les caractères spéciaux |
| n | répète la recherche |
| N | répète en inversant le sens de la recherche |

## Commandes de remplacement

|  |  |
| --- | --- |
| **Commande** | **Effet** |
| cw | Remplacement du mot courant |
| c$ | Remplacement jusqu’à la fin de la ligne |
| c0 | Remplacement jusqu'au début de la ligne |
| cfx | Remplacement jusqu’à la prochaine occurrence du caractère x, dans la ligne courante |
| c/oto | Remplacement jusqu’à la prochaine occurrence de le chaîne oto |

Note la fin de la chaîne à remplacer est marquée par le symbole $

A la fin du remplacement touche de validation enter (entrée)

## Quelques commandes du mode ex

|  |  |
| --- | --- |
| **Commande** | **Signification** |
| :wq | Quitte vi et enregistre travail sur disque dur |
| :q! | Quitte vi sans enregistrer le travail |
| :w | enregistrement du fichier sans quitter vi |
| :s/mot1/mot2 | remplace sur la ligne courante mot1 par mot2 |
| :1,5s/mot1/mot2/g | remplace de la ligne 1 à la ligne 5 mot1 par mot2 |
| :1,$s/mot1/mot2/ | remplace de la 1er à la dernière ligne mot1 par mot2 |
|  | pour la 1er occurrence dans chaque ligne |
| :.,$s/mot1/mot2/g | remplace à partir de la ligne courante à la dernière |
|  | ligne mot1 par mot2 |
| :s....../c | commande identique à g mais oblige à valider ou à rejeter le remplacement |
| :r fichier | contenu du fichier nommé est insérer dans le fichier en cours |
| :! commande | la commande est exécuté et après nous revenons à vi |
| :r! commande | contenu de la commande nommé est insérer dans le fichier en cours |
| :f | le nom du fichier courant et numéro de la ligne sont affichés |
| :e fichier | le nom du fichier indiqué est chargé si vous n’avez pas sauvegardé le fichier sur lequel vous travaillé un message vous averti |
| :e! fichier | le nom du fichier indiqué est chargé si vous n’avez pas sauvegardé le fichier sur lequel vous travaillé, il est écrasé sans message |
| :e# | le dernier fichier chargé est appelé en mémoire, permet de commuter entre plusieurs fichiers |

# Configuration de vi

## Commandes

|  |  |
| --- | --- |
| **Commande** | **Signification** |
| :set all | aperçu des divers option possible |
| :set Nom de l’option | active les options |
| :set noNom de l’option | désactive les options |
| :set option = valeur | indique une valeur à une option |

## Commutateurs

**number/nonumber**

numérotation ou non des lignes

**autoindent/noautoindent**

mise en retrait avec tabulation

annulation avec **CTRL+D**

**showmatch/noshowmatch**

pour chaque parenthèse ou accolade fermente. Positionnement sur la parenthèse ou accolade ouvrante correspondante quelque minutes puis reviens à l’emplacement initiale.

Utile pour ceux qui programment

**showmode/noshowmode**

Demande à vi d’afficher dans le coin inférieur droit le mode de travail

Command = mode commande

Input = mode saisie

(pas valable pour tous les types de terminaux)

**wrapscan/nowrapscan**

Permet de faire une espèce de boucle lors de recherche

## Options

**shiftwidth!**

Mise en retrait ou annulation des retrait de ligne.

les commandes **>>** et **<<** permettent de jouer sur des retrait de ligne complète

la largeur du retrait est déterminer par l’option **shiftwidth**

**tabstop**

un taquet de tabulation est placé toutes les 4 colonnes

l’activation de la touche tabulation vous amène directement au taquet suivant

les options présentées doivent être redéfinies à chaque session sauf si vous les enregistrez dans un fichier dans votre répertoire personnel

son nom : **.exrc**

**LES PROCESSUS** **LES PROCESSUS UNIX**

Un processus est un objet dynamique qui correspond à l'exécution d'un programme.

Le chargement d'un programme est la phase durant laquelle celui-ci passe de l'état inerte à l'état actif .

Toutes les activités dans le système (demande explicite d'utilisateur, activités purement système visant à gérer les ressources) sont exécutées dans le contexte d'un processus.

**1) On peut distinguer deux types de processus**:

**a-) les processus système** :

* ne sont sous le contrôle d'aucun terminal
* ont comme propriétaire, l'utilisateur privilégié. Ils restent résidents en mémoire centrale en attente d'une requête.
* assurent un certains nombre de services généraux accessibles à tous les utilisateurs du système ou de systèmes distants en cas d'appartenance à un réseau. Ils remplissent des tâches qui ne sont pas intégrées dans le noyau.
* peuvent être créés soit au lancement du système (ou à son passage en mode multi-utilisateurs), soit à des dates fixées par l'administrateur du système.

Exemples :

* **init** : s'assure, entre autres tâches, que sur chaque ligne de communication série asynchrone à laquelle est connecté un terminal alphanumérique, existe un processus permettant la connexion des utilisateurs (processus **getty**) et recueille les processus orphelins (problème de l'élimination des processus zombis)
* **cron** : son rôle est de lancer à des dates spécifiées des commandes particulières (exemple : effectuer une copie de sécurité des informations tous les jours à 18 heures).
* **lpshed** : son rôle est d'assurer l'ordonnancement des requêtes d'impression vers la (les) imprimante(s) que les utilisateurs soumettent via la commande **lp**.

**b ) les processus utilisateurs**.

* dédiés à l'exécution d'une tâche particulière ayant normalement une durée de vie limitée

Si la plupart des processus des utilisateurs sont sous le contrôle d'un terminal, cela n'est absolument pas une caractéristique d'un processus utilisateur .

**2) Caractéristiques d'un processus**.

Quelques caractéristiques d'un processus sont:

* son identification : PID (Processus Identifier), un nombre entier. Cet identificateur est retourné par le noyau du système lors de la création du processus. Il est unique.
* l'identificateur du processus parent: PPID, un nombre entier.
* Ses propriétaires et ses groupes propriétaires : le propriétaire et le groupe effectifs peuvent être différents du propriétaire et du groupe réels qui représentent l'utilisateur ayant lancé l'exécution de disposer d'une autre identité afin de pouvoir accéder à des fichiers dont l'accès est limité (principes "set-user-id" et "set-gid-id").
* Eventuellement son terminal d'attachement : c'est-à-dire celui à partir duquel il a été démarré.
* sa priorité : indication sur le degré d'urgence du processus. Elle permet de déterminer l'ordre d'allocation du processeur lorsque plusieurs processus sont en attente d'exécution (cas de la multi-programmation).
* TIME : temps CPU en minutes et secondes
* S ou R: état du processus Suspend ou Run

Une autre caractéristique importante des processus est qu'ils peuvent s'exécuter dans deux modes différents :

* mode utilisateur (user mode) : dans ce mode le processus n'accède qu'à son espace d'adressage et exécute des instructions ordinaires appartenant au programme chargé (par exemple: affecter à un objet z la somme des valeurs des deux objets x et y).
* mode noyau ou système (kernel mode) : le processus exécute alors des instructions qui n'appartiennent pas au programme en cours mais au noyau.

🡪accès à des données externes à son espace d'adressage (essentiellement les tables du système).

passage en mode noyau : par la demande de réalisation d'un appel système par le processus lui-même.

Exemple : la lecture de données au clavier est effectuée par une primitive du système. L'exécution du processus passe au niveau noyau pour cette phase du traitement permettant au processus d'accéder à des zones particulières de la mémoire et de communiquer avec le gestionnaire de périphériques. Après avoir reçu les informations nécessaires, l'exécution du processus continue au niveau utilisateur.

**3) Etats d'un processus et transition d'états**

Un processus en cours d'exécution, exécute les instructions qui composent le programme et accède à ses propres données. Il dialogue avec le reste des ressources disponibles par l'intermédiaire des fonctions d'appel au système.

* **un état transitoire** : à sa création ou au cours de la création d'un processus fils.
* **des états actifs** : (actif noyau ou utilisateur correspondant à son mode d'exécution) : le processus détient la ressource UC, ses propres instructions s'exécutent.
* **l'état prêt** (éligible) : le processus est éligible par l'ordonnanceur (il dispose de toutes ses ressources)
* **l'état en sommeil** : le processus est en attente d'événement (horloge, montage d'un volume disque, résultat d'une autre tâche …). Dans cet état, le processus ne consomme pas de temps CPU. Il repasse à l'état prêt si l'événement se produit.
* **l'état suspendu** : mécanisme de job control spécifique à certains interpréteurs shell comme le ksh, csh. Ce mécanisme permet aux utilisateurs de ne voir que les processus qu'ils ont lancés sous le shell interactif dans lequel ils travaillent et qui constitue le cadre d'une session de travail. Il permet également d'arrêter et de relancer des processus, de les déplacer au premier plan ou en arrière plan, de les supprimer etc…
* **l'état zombi** : le processus est terminé mais son père n'a pas pris connaissance de sa terminaison. Le processus occupe un emplacement dans la table des processus (c'est la seule ressource qu'il utilise) et ne consomme aucun temps CPU.

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

– le propriétaire (UID) ;

– le numéro du processus (PID) ;

– le numéro du processus père (PPID) ;

– la priorité (PRI) ;

– la valeur de NICE (NI) ;

– l’état (STAT)

– le nom du device vers lequel il est dirigé (TTY) ;

– le temps passé dans la cpu (TIME) ;

– le nom de la commande exécutée (COMMAND).

L’état d’un processus peut être les suivants :

– D : le processus est i interruptible ;

– R : le processus est en train de s’exécuter ;

– S : le processus est endormi ;

– T : le processus est stoppé;

– Z : le processus est zombi : il est mort mais son père ne le sait pas.

D’autres informations peuvent aussi apparaître :

– W : le processus a été swappé ;

– < : le processus possède une priorité haute ;

– N : le processus possède une priorité basse ;

– L : le processus possède des pages mémoires verrouillées.

**4) Exécution d'un processus**.

* **Commandes suivantes sont etudiées dans le chapitre shell**:

regroupements des commandes : ………(c1; c2; c3); {c1;c2;c3}

exécution conditionnelle : ………………c1 && c2 c1|| c2

communication : ………………………… c1 | c2

arrière plan :……………………………...c&

redirection ……………………………… .>, <, >>

sortir d'un processus :………………….. exit

endormir un processus : ……………….. sleep

* **Autres commandes utiles**.

a) Informations sur les processus actifs : **la commande ps**

syntaxe : **ps [options**]

La commande permet d'afficher des informations sur les processus qui appartiennent à la même session que le processus **shell** depuis lequel la commande est lancée.

Avec l'option "**–l**" et "**–f**" nous avons une liste plus complète de renseignements.

**b) la commande kill**

**kill [-signaux] numero\_processus**

La commande **kill** permet d'envoyer un signal à un processus. En général, elle est utilisée pour mettre fin à un processus.

Le ou les processus sont identifiés par leur numéro d'identification qu'il est possible de connaître à la **commande "ps".**

Seul l'administrateur a le droit de tuer des processus dont il n'est pas le propriétaire.

Le signal **SIGKILL**, de numéro " **9"**, garanti l'arrêt du processus.

**kill –9 0** met fin à tous les processus d'une session, même ceux lancé en arrière plan.

**c) processus en arrière-plan**

Lorsque l'utilisateur tape une commande, le **shell** crée un **processus fils**, le **shell** se met en attente de fin d'exécution de ce processus; c'est-à-dire que le processus **shell** est endormi et l'utilisateur n'a plus la main jusqu'à la fin de la commande.

Il est possible de ne pas endormir le processus **shell** et donc de pouvoir continuer à travailler simultanément. pour cela, il faut lancer la commande en arrière plan.

Le symbole d'exécution en arrière plan est le caractère "**&**" qui doit être ajoutée en fin de commande après les options des paramètres.

Exemple : **sort** grosfich > nouvfich **&**

./programme\_calcul\_paie **&**

Le système répond en donnant le numéro du processus **PID** ainsi créé, puis le prompt indiquant que le **shell** est à nouveau en attente de commande. L'utilisateur n'est pas averti de la fin du processus ainsi créé et si celui-ci doit afficher des messages à l'écran, il y aura affichage de ces sorties. L'utilisateur aura donc intérêt à rediriger les sorties d'une commande différée.

**d) la commande nohup**

L'utilitaire **nohup** permet à la commande passée en paramètre de continuer son exécution même si l'utilisateur se déconnecte du système (**nohup : no hangup**, pas d'arrêt imprévu).

Si une demande de redirection de la sortie standard n'a pas été formulée, celle-ci et la sortie erreur standard sont automatiquement redirigées sur le fichier **nohup.out** du répertoire de travail.

# Exemple

**nohup sort** grosfich **>** nouvfich **&**

**e) suspension d’un processus**

Si un processus est exécuté en avant plan, il est possible de le suspendre en frappant simultanément sur les touches **Ctrl -Z** . Un processus suspendu peut être redémarré en avant-plan via la commande **fg** (pour foreground) ou en arrière plan avec la commande **bg** (pour background).

**SHELL** **LE SHELL ET SHELL-SCRIPTS**

**1 - PRESENTATION**

Le **shell** est l’interpréteur de commandes d’ Unix.

Double fonction :

* En mode interactif, il constitue l interface des utilisateurs avec le système.
* A un **shell** est associé un véritable langage de programmation disposant de mécanismes internes permettant en particulier l’écriture de nouvelles commandes (scripts **shell** ).

**Démarrage du shell**.

* lors de la création de son compte, un utilisateur est associé à un type de **shell .** C'est le nom du fichier exécutable qui est situé dans le dernier champ de la ligne définissant l'utilisateur dans le fichier **/etc/passwd (ex : /bin/ksh)**
* Le **shell** associé est ainsi lancé automatiquement dès la saisie du login utilisateur.
* Il affiche le prompt et se met en attente de la lecture d'une commande jusqu'à la commande exit (ou **CTRL-D**), pour quitter le **shell** (ce qui équivaut à se déconnecter (logout))

**Syntaxe général d’une commande.**

***nom\_de\_commande*** [-option(s)] [paramètre(s)]

Une commande est une séquence de mots séparés par des caractères spéciaux, le premier mot est le nom de la commande à exécuter, les suivants étant les paramètres.

Voici la liste des caractères spéciaux pouvant appartenir à un pattern et leur équivalence :

Qu'est-ce qui rend un caractère *spécial* ? S'il a une signification en dehors de la *signification littérale*, une meta signification, alors nous lui donnons le nom de *caractère spécial*.

**SHELL**

**LES CARACTERES SPECIAUX DU SHELL**

**2 CARACTERES SPECIAUX**

**Le caractère #**

**Commentaires.** Les lignes commençant avec un # (à l'exception de #!) sont des commentaires et ne seront pas *exécutées*.

# Cette ligne est un commentaire.

Les commentaires peuvent apparaître après la fin d'une commande.

echo "Un commentaire va suivre." # Un commentaire ici.

# ^ Notez l'espace blanc devant #

Les commentaires peuvent aussi suivre un blanc au début d'une ligne.

# Une tabulation précède ce commentaire.

**Attention**

Un commentaire ne peut pas être suivi d'une commande sur la même ligne. Il n'existe pas de façon de terminer

le commentaire pour que le « vrai code » commence sur la même ligne. Utilisez une nouvelle ligne pour la

commande suivante.

**Note**

Bien sûr, un " ou un # dans une instruction echo ne commence *pas* un commentaire. De la même manière, un # apparaît dans certaines constructions de substitution de paramètres et dans les expressions numériques constantes.

echo "Le # ici ne commence pas un commentaire."

echo 'Le # ici ne commence pas un commentaire.'

echo Le \# ici ne commence pas un commentaire.

echo Le # ici commence un commentaire.

echo ${PATH#\*:} # Substitution de paramètres, pas un commentaire.

echo $(( 2#101011 )) # Conversion de base, pas un commentaire.

Les caractères standards (" ' \) échappent le #. Certaines opérations de filtrage de motif font aussi appel au #.

**Le caractère ;**

**Séparateur de commande [point-virgule].** Permet de placer deux commandes ou plus sur la même ligne.

echo bonjour; echo ici

if [ -x "$nomfichier" ]; then # Notez que "if" et "then" doivent être séparés# par un espace blanc. Pourquoi ?

echo "Le fichier $nomfichier existe."; cp $nomfichier $nomfichier.sauve

else

echo "Le fichier $nomfichier est introuvable."; touch $nomfichier

fi; echo "Test du fichier terminé."

**Les caractères ;;**

**Fin de ligne dans une sélection par cas case [double point-virgule].**

case "$variable" in

abc) echo "\$variable = abc" ;;

xyz) echo "\$variable = xyz" ;;

esac

**Le caractère .**

**Commande « point » [point].** Équivalent à la source, « « Inclure » un fichier de données ». C'est une

commande intégrée de Bash.

.

**« point », comme composant d'un nom de fichier.** Lors de l'utilisation de noms de fichiers, un point au début est le préfixe d'un fichier « caché », un fichier que ls ne montre habituellement pas.

bash$ **touch .fichier\_caché**

bash$ **ls -l**

total 10

-rw-r--r-- 1 bozo bozo 4034 Jul 18 22:04 donnée1.carnet\_d\_adresses

-rw-r--r-- 1 bozo bozo 4602 May 25 13:58 donnée1.carnet\_d\_adresses.bak

-rw-r--r-- 1 bozo bozo 877 Dec 17 2000 boulot.carnet\_d\_adresse

bash$ **ls -al**

total 14

drwxrwxr-x 2 bozo bozo 1024 Aug 29 20:54 ./

drwx------ 52 bozo bozo 3072 Aug 29 20:51 ../

-rw-r--r-- 1 bozo bozo 4034 Jul 18 22:04 donnée1.carnet\_d\_adresses

-rw-r--r-- 1 bozo bozo 4602 May 25 13:58 donnée1.carnet\_d\_adresses.bak

-rw-r--r-- 1 bozo bozo 877 Dec 17 2000 boulot.carnet\_d\_adresse

-rw-rw-r-- 1 bozo bozo 0 Aug 29 20:54 .fichier\_caché

En ce qui concerne les noms des répertoires, *un seul point* représente le répertoire courant et *deux points de suite* indiquent le répertoire parent.

bash$ **pwd**

/home/bozo/projets

bash$ **cd .**

bash$ **pwd**

/home/bozo/projets

bash$ **cd ..**

bash$ **pwd**

/home/bozo/

Le *point* apparaît souvent comme répertoire de destination d'une commande de mouvement de fichiers. Dans ce contexte, cela signifie le *répertoire courant.*.

bash$ **cp /home/bozo/travail\_en\_cours/débarras/\* .**

Copiez tous les fichiers du répertoire « débarras » dans $PWD.

.

**Filtrage d'un caractère par le « point ».** Pour le filtrage de caractères au sein d'une expression rationnelle, un « point » correspond à un seul caractère.

**Le caractère "**

**Citation partielle [guillemet double].***"CHAÎNE"* empêche l'interprétation de la plupart des caractères spéciaux présents dans la *CHAÎNE.*

Un *opérateur* est un agent qui exécute une *opération*. L'exemple habituel est l'opérateur arithmétique, **+ - \* /**. Avec Bash, il y a croisement entre les concepts d'*opérateur* et de mots clés.

**Le caractère '**

**Citation totale [guillemet simple].** *'CHAÎNE'* empêche l'interprétation de tous les caractères spéciaux présents dans la *CHAÎNE*. Ces guillemets sont plus puissants que *"CHAÎNE"*.

**Le caractère ,**

**Opérateur virgule.** L'*opérateur virgule*  relie une suite d'opérations arithmétiques. Toutes sont évaluées, mais seul le résultat de la dernière est renvoyé.

let "t2 = ((a = 9, 15 / 3))" # Initialise "a = 9" et "t2 = 15 / 3".

**Le caractère \**

**Échappement39 [antislash].**Un mécanisme d'échappement pour les caractères seuls.

**\X** *échappe* le caractère *X*. Cela a pour effet de « mettre *X* entre guillemets », et est équivalent à *'X'*. Le \ peut être utilisé pour mettre " et ' entre guillemets, ce qui permet de les écrire sous forme littérale.

**Le caractère /**

**Séparateur dans le chemin d'un fichier [barre oblique].** Sépare les composants d'un nom de fichier (comme dans /home/bozo/projets/Makefile).

C'est aussi l'opérateur arithmétique de division.

**Le caractère `**

**Substitution de commandes [guillemet inversé]. L**a construction **`commande`** rend la sortie de **commande** disponible pour l'affecter à une variable. Connu sous le nom de guillemets inversés.

**Le caractère :**

**Commande nul [deux-points].** Il s'agit de l'équivalent shell d'un « NOP » (*no op*, c'est-à-dire « pas d'opération »). Elle peut être considérée comme un synonyme pour la commande intégrée true.

La commande « : » est elle-même une commande intégrée *Bash* et son état de sortie est *true* (0).

:

echo $? # 0

Boucle sans fin :

while :

do

operation-1

operation-2

...

operation-n

done

# Identique à :

# while true

# do

# ...

# done

Sert de bouche-trou dans un test if/then :

if condition

then : # Ne rien faire et continuer

else

faire\_quelque\_chose

fi

Sert de bouche-trou quand on attend une opération binaire, voir l'Exemple 8.2, « Utiliser des opérations arithmétiques » et les paramètres par défaut.

: ${nom\_utilisateur=`whoami`}

# ${nom\_utilisateur=`whoami`} donne une erreur sans les deux-points en tout début

# sauf si "nom\_utilisateur" est une commande, intégrée ou non...

Évalue une suite de variables en utilisant la substitution de paramètres ( « Utiliser la substitution et les messages d'erreur »).

: ${HOSTNAME?} ${USER?} ${MAIL?}

# Affiche un message d'erreur

#+ si une variable d'environnement (ou plusieurs) n'est pas initialisée.

**Expansion de variable / remplacement d'une sous-chaîne**.

En combinaison avec l'opérateur de redirection >, tronque un fichier à la taille zéro sans modifier ses droits.

Crée le fichier s'il n'existait pas auparavant.

: > données.xxx # Fichier "données.xxx" maintenant vide

# Même effet que cat /dev/null >données.xxx

# Néanmoins, cela ne crée pas un nouveau processus, car ":" est une commande intégrée.

En combinaison avec l'opérateur de redirection >>, elle n'a pas d'effet sur un fichier cible déjà existant (**: >> nouveau\_fichier**). Crée le fichier s'il n'existait pas.

**Note**

Cela s'applique aux fichiers réguliers, mais pas aux tubes, aux liens symboliques et à certains fichiers spéciaux.

Peut servir à commencer une ligne de commentaire bien que ce ne soit pas recommandé. Utiliser # pour un commentaire désactive la vérification d'erreur pour le reste de la ligne, donc vous pouvez y mettre pratiquement n'importe quoi. En revanche, ce n'est pas le cas avec :.

: Ceci est un commentaire qui génère une erreur, ( if [ $x -eq 3] ).

Le « : » sert aussi de séparateur de champ, dans /etc/passwd et dans la variable $PATH.

bash$ **echo $PATH**

/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/sbin:/usr/sbin:/usr/games

**Le caractère !**

**Inverse le sens d'un test ou d'un état de sortie.** L'opérateur ! inverse l'état de sortie de la commande à laquelle il est appliqué. Il inverse aussi la signification d'un opérateur de test. Par

exemple, cela peut changer le sens d'un *égal* (=) en un *différent* ( != ). L'opérateur ! est un mot-clé Bash.

Dans un autre contexte, le ! apparaît aussi dans les références indirectes de variable.

Dans un contexte encore différent, à partir de la *ligne de commande*, le ! appelle le *mécanisme d'historique* de Bash. Notez que ce mécanisme est désactivé dans les scripts.

**Le caractère \***

**Joker [astérisque].** Le caractère \* sert de « joker » pour l'expansion des noms de fichiers dans le remplacement. Utilisé seul,

il correspond à tous les noms de fichiers d'un répertoire donné.

bash$ **echo \***

abs-book.sgml add-drive.sh agram.sh alias.sh

L'astérisque \* représente tout caractère répété plusieurs fois (ou zéro) dans une expression rationnelle.

**Opérateur arithmétique.** Dans le contexte des opérations arithmétiques, \* indique la multiplication.

Le double astérisque \*\* indique l'opérateur exponentiel.

**Le caractère ?**

**Opérateur de test.** À l'intérieur de certaines expressions, le ? indique un test pour une condition.

Dans une construction entre parenthèses doubles, ? peut servir en tant qu'élément d'opérateur *à trois arguments* dans le style

du C, ?:.

(( var0 = var1<98?9:21 ))

# ^ ^

# if [ "$var1" -lt 98 ]

# then

# var0=9

# else

# var0=21

# fi

Dans une expression de substitution de paramètres, le ? teste si une variable a été initialisée.

? **Joker.** Le caractère ? sert de joker pour un seul caractère dans l'expansion d'un nom de fichier dans un remplacement, et représente également un caractère dans une expression rationnelle étendue.

**Le caractère $**

**Substitution de variable (contenu d'une variable).**

var1=5

var2=23skidoo

echo $var1 # 5

echo $var2 # 23skidoo

Un $ préfixant un nom de variable donne la *valeur* que contient cette variable.

$ **Fin de ligne.** Dans une expression rationnelle, un $ signifie la fin d'une ligne de texte.

${} **Substitution de paramètres.**

$\*, $@ **Paramètres de position.**

? **Variable contenant l'état de sortie.** La variable $? contient l'état de sortied'une commande, d'une fonction ou d'un script.

$$ **Variable contenant l'identifiant du processus.** La variable $$ contient le *PID*  du script dans lequel elle apparaît.

**Les caractères ()**

**Groupe de commandes.**

(a=bonjour; echo $a)

**Important**

Une liste de commandes entre *parenthèses* lance un sous-shell.Les variables comprises dans ces parenthèses, à l'intérieur du sous-shell, ne sont pas visibles par le reste du script. Le processus parent, le script, ne peut pas lire les variables créées dans le processus fils, le sous-shell.

a=123

( a=321; )

echo "a = $a" # a = 123

# "a" à l'intérieur des parenthèses agit comme une variable locale

**Initialisation de tableaux.**

Tableau=(element1 element2 element3)

**{xxx, yyy, zzz, ...} Expansion d'accolades.**

cat {fichier1,fichier2,fichier3} > fichier\_combiné

# Concatène les fichiers fichier1, fichier2 et fichier3 dans fichier\_combiné.

cp fichier22.{txt,sauve}

# Copie "fichier22.txt" dans "fichier22.sauve"

Une commande peut agir sur une liste de fichiers séparés par des virgules entre des *accolades*. L'expansion de noms de fichiers (remplacement) s'applique aux fichiers contenus dans les accolades.

**Attention**

Aucune espace n'est autorisée à l'intérieur des accolades *sauf si* les espaces sont comprises dans des guillemets

ou échappés.

**echo {fichier1,fichier2}\ :{\ A," B",' C'}**

fichier1 : A fichier1 : B fichier1 : C fichier2 : A fichier2 : B fichier2

: C

**{a..z} Expansion étendue d'accolades.**

echo {a..z} # a b c d e f g h i j k l m n o p q r s t u v w x y z

# Affiche les caractères entre a et z.

echo {0..3} # 0 1 2 3

# Affiche les caractères entre 0 et 3.

**{} Bloc de code [accolade].** Aussi connu sous le nom de *groupe en ligne*, cette construction crée une *fonction anonyme* (une fonction sans nom). Néanmoins, contrairement à une fonction standard, les variables d'un bloc de code restent visibles par le reste du script.

bash$ **{ local a;**

**a=123; }**

bash: local: can only be used in a function

a=123

{ a=321; }

echo "a = $a" # a = 321 (valeur à l'intérieur du bloc de code)

Le bloc de code entouré par des accolades peut utiliser la redirection d'entrées/sorties.

**Exemple 3.1. Blocs de code et redirection d'entrées/sorties**

#!/bin/bash

# Lit les lignes de /etc/fstab.

Fichier=/etc/fstab

{

read ligne1

read ligne2

} < $Fichier

echo "La première ligne dans $Fichier est :"

echo "$ligne1"

echo

echo "La deuxième ligne dans $Fichier est :"

echo "$ligne2"

exit 0

# Maintenant, comment analysez-vous les champs séparés de chaque ligne ?

# Astuce : utilisez awk, ou...

# ... Hans-Joerg Diers suggère d'utiliser la commande set de Bash.

**Exemple 3.2. Sauver la sortie d'un bloc de code dans un fichier**

#!/bin/bash

# rpm-check.sh

# Recherche une description à partir d'un fichier rpm, et s'il peut être

#+ installé.

# Sauvegarde la sortie dans un fichier.

##

Ce script illustre l'utilisation d'un bloc de code.

SUCCES=0

E\_SANSARGS=65

if [ -z "$1" ]

then

echo "Usage: `basename $0` fichier-rpm"

exit $E\_SANSARGS

fi

{ # Début du bloc de code

echo

echo "Description de l'archive :"

rpm -qpi $1 # Requête pour la description.

echo

echo "Contenu de l'archive :"

rpm -qpl $1 # Requête pour la liste.

echo

rpm -i --test $1 # Requête pour savoir si le fichier rpm est installable.

if [ "$?" -eq $SUCCES ]

then

echo "$1 est installable."

else

echo "$1 n'est pas installable."

fi

echo # Fin du bloc de code

} > "$1.test" # Redirige la sortie de tout le bloc vers un fichier.

echo "Les résultats du test rpm sont dans le fichier $1.test"

# Voir la page de manuel de rpm pour des explications sur les options.

Exception : un bloc de code entre accolades dans un tube *peut* être lancé comme sous-shell.

ls | { read ligne1; read ligne2; }

# Erreur. Le bloc de code entre accolades tourne comme un sous-shell,

#+ donc la sortie de "ls" ne peut être passée aux variables de ce bloc.

echo "La première ligne est $ligne1; la seconde ligne est $ligne2" # Ne fonctionnera pas.

exit 0

**Note**

Contrairement à un groupe de commandes entre parenthèses, comme ci-dessus, un bloc de code entouré par

des accolades ne sera *pas* lancé dans un sous-shell.

**{} Emplacement pour du texte.**

Utilisé après xargs –i (option de*replacement de chaîne*). Les doubles accolades {} sont un emplacement pour du texte en sortie.

ls . | xargs -i -t cp ./{} $1

# ^^ ^^

# Provient de l'exemple "ex42.sh" (copydir.sh).

{} \;

**Chemin.** Principalement utilisé dans les constructions find. Ce n'est *pas* une commande intégrée du shell.

**Note**

Le « ; » termine l'option -exec d'une séquence de commandes **find**. Il a besoin d'être échappé pour que le

shell ne l'interprète pas.

**[ ] Test.**

Teste l'expression entre**[ ]**. Notez que **[** fait partie de la commande *intégrée* test (et en est un synonyme), ce n'est *pas* un lien vers la commande externe /usr/bin/test.

[[ ]] **Test.**

Teste l'expression entre [[ ]]. C'est un mot-clé du shell.

[ ] **Élément d'un tableau.**

Accolés au nom d'un tableau, les crochets indiquent l'indice d'un élément.

Tableau[1]=slot\_1

echo ${Tableau[1]}

[ ] **Ensemble de caractères.**

Dans une expression rationnelle, les crochets désignent un ensemble de caractères devant servir de motif (N.d.T : cet ensemble peut être un intervalle).

**(( )) Expansion d'entiers.**

Développe et évalue une expression entière entre (( )).

Voir les explications sur la structure (( ... )).

**> &> >& >> < <> Redirection.**

**nom\_script >nom\_fichier** redirige la sortie de nom\_script vers le fichier nom\_fichier et écrase nom\_fichier s'il existe déjà.

**commande &>nom\_fichier** redirige à la fois stdout esttderr de commande vers nom\_fichier.

**commande >&2** redirige stdout de commande vers stderr.

**nom\_script >>nom\_fichier** ajoute la sortie de nom\_script à la fin du fichier nom\_fichier. Si le fichier n'existe

pas déjà, il est créé.

**[i]<>nom\_fichier** ouvre le fichier nom\_fichier en lecture et écriture, et lui affecte le descripteur de fichier i. Si

nom\_fichier n'existe pas, il est créé.

**Substitution de processus.**

**(commande)>**

**<(commande)**

Dans un autre contexte, les caractères < et > agissent comme des opérateurs de comparaison de chaînes de caractères.

Dans un contexte encore différent, les caractères < et > agissent comme des opérateurs de comparaison d'entiers.

<< **Redirection utilisée dans un document en ligne.**

<<< **Redirection utilisée dans une chaîne en ligne.**

<, > **Comparaison ASCII.**

leg1=carottes

leg2=tomates

if [[ "$leg1" < "$leg2" ]]

then

echo -n "Le fait que $leg1 précède $leg2 dans le dictionnaire "

echo "n'a rien à voir avec mes préférences culinaires."

else

echo "Mais quel type de dictionnaire utilisez-vous?"

fi

\<, \> **Délimitation d'un mot314 dans une expression rationnelle313.**

bash$ **grep '\<mot\>' fichier\_texte**

**Le caractère | Tube.**

Passe la sortie (stdout) de la commande précédente à l'entrée (stdin) de la suivante ou au shell. Cette méthode permet de chaîner les commandes ensemble.

echo ls -l | sh

# Passe la sortie de "echo ls -l" au shell

#+ avec le même résultat qu'un simple "ls -l".

cat \*.lst | sort | uniq

# Assemble et trie tous les fichiers ".lst", puis supprime les lignes

#+ dupliquées.

Un tube, méthode classique de communication inter-processus, envoie le canal stdout d'un processus au canal stdin d'un autre processus. Dans un cas typique, une commande, comme cat ou echo, envoie un flux de données à un *filtre* (une autre commande) qui opérera des transformations sur ces données.

**cat $nom\_fichier1 $nom\_fichier2 | grep $mot\_recherché**

La *partie 3 de la FAQ UNIX* contient une note intéressante sur la complexité de l'utilisation de tubes UNIX.

La sortie d'une ou plusieurs commandes peut être envoyée à un script via un tube.

#!/bin/bash

# uppercase.sh : Change l'entrée en majuscules.

tr 'a-z' 'A-Z'

# La plage de lettres doit être entre guillemets pour empêcher que la

#+ génération des noms de fichiers ne se fasse que sur les fichiers à un

#+ caractère.

exit 0

Maintenant, envoyons par le tube la sortie de **ls -l** à ce script.

bash$ **ls -l | ./uppercase.sh**

-RW-RW-R-- 1 BOZO BOZO 109 APR 7 19:49 1.TXT

-RW-RW-R-- 1 BOZO BOZO 109 APR 14 16:48 2.TXT

-RW-R--R-- 1 BOZO BOZO 725 APR 20 20:56 FICHIER-DONNEES

**Note**

Le canal stdout de chaque processus dans un tube doit être lu comme canal stdin par le suivant. Si ce n'est

pas le cas, le flux de données va se *bloquer* et le tube ne se comportera pas comme il devrait.

cat fichier1 fichier2 | ls -l | sort

# La sortie à partir de "cat fichier1 fichier2" disparaît.

Un tube tourne en tant que processus fils et ne peut donc modifier les variables du script.

variable="valeur\_initiale"

echo "nouvelle\_valeur" | read variable

echo "variable = $variable" # variable = valeur\_initiale

Si une des commandes du tube échoue, l'exécution du tube se termine prématurément. Dans ces conditions, on

a un *tube cassé* et on envoie un signal *SIGPIPE*.

**Les caractères >|**

**Force une redirection (même si l' option noclobber est activée).** Ceci va forcer l'écrasement d'un fichier déjà existant.

**Les caractères |**

**|**

**Opérateur logique OU.** Dans une structure de test , l'opérateur || a comme valeur de retour 0 (succès) si *l'une des conditions* est vraie.

**Le caractère &**

**Exécuter la tâche en arrière-plan.** Une commande suivie par un & fonctionnera en tâche de fond.

bash$ **sleep 10 &**

[1] 850

[1]+ Done sleep 10

À l'intérieur d'un script, les commandes et même les boucles peuvent tourner en tâche de fond.

**Exemple 3.3. Exécuter une boucle en tâche de fond**

#!/bin/bash

# background-loop.sh

for i in 1 2 3 4 5 6 7 8 9 10 # Première boucle.

do

Caractères spéciaux

19

echo -n "$i "

done & # Exécute cette boucle en tâche de fond.

# S'exécutera quelques fois après la deuxième boucle.

echo # Ce 'echo' ne s'affichera pas toujours.

for i in 11 12 13 14 15 16 17 18 19 20 # Deuxième boucle.

do

echo -n "$i "

done

echo # Ce 'echo' ne s'affichera pas toujours.

# ======================================================

# La sortie attendue de ce script :

# 1 2 3 4 5 6 7 8 9 10

# 11 12 13 14 15 16 17 18 19 20

# Mais, quelque fois, vous obtenez :

# 11 12 13 14 15 16 17 18 19 20

# 1 2 3 4 5 6 7 8 9 10 bozo $

# (Le deuxième 'echo' ne s'exécute pas. Pourquoi ?)

# Occasionnellement aussi :

# 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

# (Le premier 'echo' ne s'exécute pas. Pourquoi ?)

# Et très rarement :

# 11 12 13 1 2 3 4 5 6 7 8 9 10 14 15 16 17 18 19 20

# La boucle en avant plan s'exécute avant celle en tâche de fond.

exit 0

# Nasimuddin Ansari suggère d'ajouter sleep 1

#+ après le echo -n "$i" aux lignes 6 et 14,

#+ pour un peu d'amusement.

**Attention**

Une commande exécutée en tâche de fond à l'intérieur d'un script peut faire se suspendre l'exécution, attendant

l'appui sur une touche. Heureusement, il est possible d'y remédier.

**Les caractères &&**

**Opérateur logique ET.** Dans une structure de test, l'opérateur && renvoie 0 (succès) si et seulement si les *deux* conditions sont vraies.

**Le caractère –**

**Option, préfixe.** Introduit les options pour les commandes ou les filtres. Sert aussi de préfixe pour un opérateur. Préfixe pour un paramètre par défaut dans la substitution de paramètres.

COMMANDE -[Option1][Option2][...]

ls -al

sort -dfu $nom\_fichier

if [ $fichier1 -ot $fichier2 ]

then # ^

echo "Le fichier $fichier1 est plus ancien que le $fichier2."

fi

if [ "$a" -eq "$b" ]

then

echo "$a est égal à $b."

fi

Caractères spéciaux

20

if [ "$c" -eq 24 -a "$d" -eq 47 ]

then

echo "$c vaut 24 et $d vaut 47."

fi

param2=${param1:-$DEFAULTVAL}

**Les caractères –**

Le *tiret double* -- est le préfixe des options *longues* pour les commandes.

**sort --ignore-leading-blanks**

Utilisé avec une commande interne Bash, il signifie la *fin des options* de cette commande spécifique.

**Astuce**

Ceci donne un moyen simple pour supprimer les fichiers dont *les noms commencent avec un tiret*.

bash$ **ls -l**

-rw-r--r-- 1 bozo bozo 0 Nov 25 12:29 -mauvaisnom

bash$ **rm -- -mauvaisnom**

bash$ **ls -l**

total 0

Le *double-tiret* est aussi utilisé avec set.

set -- $variable

Redirection à partir de ou vers stdin ou stdout [tiret].

bash$ cat -

abc

abc

...

Ctrl D

Comme attendu, **cat -** lit sur stdin, dans ce cas la saisie de l'utilisateur au clavier, et envoie vers stdout. Mais, est-ce que la redirection des entrées/sorties utilisant **-** ont une application réelle ?

(cd /source/répertoire && tar cf - . ) | (cd /dest/répertoire && tar xpvf -)

# Déplace l'ensemble des fichiers d'un répertoire vers un autre

# 1) cd /source/répertoire

# Répertoire source, où se trouvent les fichiers à déplacer.

# 2) &&

# "liste ET": si l'opération 'cd' a fonctionné,

# alors il exécute la commande suivante.

# 3) tar cf - .

# L'option 'c' de la commande d'archivage 'tar' crée une nouvelle archive,

# l'option 'f' (fichier), suivie par '-' désigne stdout comme fichier cible.

# et place l'archive dans le répertoire courant ('.').

# 4) |

# Tube...

# 5) ( ... )

# Un sous-shell.

# 6) cd /dest/répertoire

# Se déplace dans le répertoire de destination.

Caractères spéciaux

21

# 7) &&

# "liste ET", comme ci-dessus.

# 8) tar xpvf -

# Déballe l'archive ('x'), préserve l'appartenance

# et les droits des fichiers ('p'),

# puis envoie de nombreux messages vers stdout ('v'),

# en lisant les données provenant de stdin

# ('f' suivi par un '-').

##

Notez que 'x' est une commande, et 'p', 'v', 'f' sont des options.

# Plus élégant, mais équivalent à :

# cd /source/répertoire

# tar cf - . | (cd ../dest/répertoire; tar xpvf -)

##

A aussi le même effet :

# cp -a /source/répertoire/\* /dest/répertoire

# Ou :

# cp -a /source/répertoire/\* /source/répertoire/.[^.]\* /dest/répertoire

# S'il y a des fichiers cachés dans /source/répertoire.

bunzip2 -c linux-2.6.16.tar.bz2 | tar xvf -

# --décompresse l'archive-- | --puis la passe à "tar"--

# Si "tar" n'a pas intégré le correctif de support de "bunzip2",

#+ il faut procéder en deux étapes distinctes avec un tube.

# Le but de cet exercice est de désarchiver les sources du noyau compressées

#+ avec bzip2.

Notez que dans ce contexte le signe « - » n'est pas en lui-même un opérateur Bash, mais plutôt une option reconnue par certains utilitaires UNIX qui écrivent dans stdout ou lisent dans stdin, tels que **tar**, **cat**, etc.

bash$ **echo "quoiquecesoit" | cat -**

quoiquecesoit

Lorsqu'un nom de fichier est attendu, un *-* redirige la sortie vers stdout (vous pouvez le rencontrer avec **tar cf**), ou accepte une entrée de stdin, plutôt que d'un fichier. C'est une méthode pour utiliser un outil principalement destiné à manipuler des fichiers comme filtre dans un tube.

bash$ **file**

Usage: file [-bciknvzL] [-f namefile] [-m magicfiles] file...

Tout seul sur la ligne de commande, file échoue avec un message d'erreur.

Ajoutez un « - » pour pouvoir vous en servir. Le shell attend alors une entrée de l'utilisateur.

bash$ **file -**

**abc**

standard input: ASCII text

bash$ **file -**

**#!/bin/bash**

standard input: Bourne-Again shell script text executable

Maintenant, la commande accepte une entrée de stdin et l'analyse.

Le « - » peut être utilisé pour envoyer stdout à d'autres commandes via un tube, ce qui permet quelques astuces comme l'ajout de lignes au début d'un fichier.

Par exemple, vous pouvez utiliser diff pour comparer un fichier avec une *partie* d'un autre fichier :

**grep Linux fichier1 | diff fichier2 -**

Finalement, un exemple réel utilisant *-* avec tar.

**Exemple 3.4. Sauvegarde de tous les fichiers modifiés dans les dernières 24 heures**

#!/bin/bash

# Sauvegarde dans une archive tar compressée tous les fichiers

#+ du répertoire courant modifiés dans les dernières 24 heures.

FICHIERSAUVE=backup-$(date +%m-%d-%Y)

# Intégration de la date dans le nom du fichier de sauvegarde.

# Merci pour cette idée, Joshua Tschida.

archive=${1:-$FICHIERSAUVE}

# Si aucun nom de fichier n'est spécifié sur la ligne de commande,

#+ nous utiliserons par défaut "backup-MM-JJ-AAAA.tar.gz."

tar cvf - `find . -mtime -1 -type f -print` > $archive.tar

gzip $archive.tar

echo "Répertoire $PWD sauvegardé dans un fichier archive \"$archive.tar.gz\"."

# Stephane Chazelas indique que le code ci-dessus échouera si il existe trop

#+ de fichiers ou si un nom de fichier contient des espaces blancs.

# Il suggère les alternatives suivantes:

# -------------------------------------------------------------------

# find . -mtime -1 -type f -print0 | xargs -0 tar rvf "$archive.tar"

# avec la version GNU de "find".

# find . -mtime -1 -type f -exec tar rvf "$archive.tar" '{}' \;

# portable aux autres UNIX, mais plus lent.

# -------------------------------------------------------------------

exit 0

**Attention**

Les noms de fichiers commençant avec un « - » peuvent poser problème lorsqu'ils sont couplés avec

l'opérateur de redirection « - ». Votre script doit détecter de tels fichiers et leur ajouter un préfixe approprié,

par exemple ./-NOMFICHIER, $PWD/-NOMFICHIER, ou $NOMCHEMIN/-NOMFICHIER.

Il y aura probablement un problème si la valeur x d'une variable commence avec un *-*.

var="-n"

echo $var

# A le même effet qu'un "echo -n" et donc n'affiche rien.

**Les caractères d’opération**

**Répertoire précédent. cd -** revient au répertoire précédent en utilisant la variable d'environnement $OLDPWD.

**Attention**

Ne confondez pas « - » utilisé dans ce sens avec l'opérateur de redirection « - » vu précédemment.

L'interprétation du « - » dépend du contexte dans lequel il apparaît.

**- Moins.** Le signe moins indique l'opération arithmétique.

= **Égal.** Opérateur d'affectation.

a=28

echo $a # 28

Dans un autre contexte, le signe = est un opérateur de comparaison de chaînes de caractères.

+ **Plus.** Opérateur arithmétique d'addition.

Dans un autre contexte315, le + est un opérateur d'expression rationnelle.

+ **Option.** Option pour une commande ou un filtre.

Certaines commandes, intégrées159 ou non, utilisent le + pour activer certaines options et le - pour les désactiver. Dans la substitution

de paramètres, le + préfixe une autre valeur qu'une variable étend.

% **Modulo.** Opérateur arithmétique modulo (reste d'une division entière).

let "z = 5 % 3"

echo $z # 2

Dans un autre contexte, le % est un opérateur de reconnaissance de motifs.

**Le caractère ~**

**Répertoire de l'utilisateur [tilde].** Le ~ correspond à la variable interne $HOME. *~bozo* est le répertoire de l'utilisateur bozo et **ls ~bozo** liste son contenu. ~/ est le répertoire de l'utilisateur courant et **ls ~/** liste son contenu.

bash$ **echo ~bozo**

/home/bozo

bash$ **echo ~**

/home/bozo

bash$ **echo ~/**

/home/bozo/

bash$ **echo ~:**

/home/bozo:

bash$ **echo ~utilisateur-inexistant**

~utilisateur-inexistant

~+ **Répertoire courant.** Correspond à la variable interne $PWD.

~-**Répertoire courant précédent.** Correspond à la variable interne $OLDPWD.

=~ **correspondance d'une expression rationnelle** C**.** et opérateur

**Début de ligne.** Dans une expression rationnelle, un « ^ » correspond au début d'une ligne de texte.

**Les caractères de contrôle.**

**Modifient le comportement d'un terminal ou de l'affichage d'un texte.** Un caractère de contrôle est une combinaison **CONTROL** + **touche** (appuyés simultanément). Un caractère de contrôle peut aussi être écrit en notation *octal* ou *hexadécimal*, après un *échappement*.

Les caractères de contrôle ne sont normalement pas utiles à l'intérieur d'un script.

• **Ctl-A** Déplace le curseur au début de la ligne de texte (sur la ligne de commande).

• **Ctrl-B** Retour en arrière (*backspace*) non destructif.

• **Ctrl-C** Termine un job en avant-plan.

• **Ctrl-D** *Se déconnecte* du shell (similaire à un exit44).

C'est le caractère **EOF** (End Of File, fin de fichier), qui termine aussi l'entrée de stdin.

Lors de la saisie de texte sur la console ou dans une fenêtre *xterm*, **Ctl-D** efface le caractère sous le curseur. Quand aucun caractère n'est présent, **Ctl-D** vous déconnecte de la session. Dans une fenêtre xterm, ceci a pour effet de fermer la fenêtre.

• **Ctl-E** Moves cursor to end of line of text (on the command-line).

• **Ctl-F** Moves cursor forward one character position (on the command-line).

• **Ctrl-G** (bip). Sur quelques anciens terminaux comme les télétypes, ceci fera vraiment sonner une cloche. Dans un *xterm*, cela pourrait sonner.

• **Ctrl-H** Supprime le caractère précédent (*Backspace*). Efface les caractères sur lequel le curseur passe en arrière.

#!/bin/bash

# Intègre Ctrl-H dans une chaîne de caractères.

a="^H^H" # Deux Ctrl-H (backspaces)

# Ctrl-V Ctrl-H si vous utilisez vi/vim

echo "abcdefg" # abcdefg

echo

echo -n "abcdefg$a " # abcd fg

# Espace à la fin ^ ^ Deux fois backspaces.

echo

echo -n "abcdef$a" # abcdef

# Pas d'espace à la fin ^ Ne fait pas de backspace (pourquoi?).

# Les résultats pourraient ne pas être ceux attendus.

echo; echo

# Constantin Hagemeier suggère d'essayer :

# a=$'\010\010'

# a=$'\b\b'

# a=$'\x08\x08'

# mais cela ne modifie pas les résultats.

• **Ctrl-I Tabulation horizontale**.

• **Ctrl-J Saut à la ligne** (*line feed*). Dans un script, cela pourrait aussi s'exprimer en notation octale -- '\012' ou en notation hexadécimal -- '\x0a'.

• **Ctrl-K Tabulation verticale**.

Lors de la saisie de texte sur la console ou dans une fenêtre *xterm*, **Ctl-K** efface les caractères en commençant à partir du

curseur jusqu'à la fin de la ligne. Within a script, **Ctl-K** may behave differently, as in Lee Lee Maschmeyer's example,

below.

• **Ctrl-L Formfeed** (efface l'écran du terminal). Dans un terminal, ceci a le même effet que la commande clear. Une fois envoyé à une imprimante, un **Ctl-L** éjecte la page de papier.

Bash stocke une liste de commandes auparavant lancées à partir de la ligne de commande dans un *tampon*, ou dans un espace en mémoire, pour retrouver l'historique à partir des commandes internes de l'*historique*.

• **Ctrl-M Retour chariot**.

#!/bin/bash

read -n 1 -s -p \

$'Control-M place le curseur au début de cette ligne. Tapez sur Enter. \x0d'

# Bien sûr, '0d' est l'équivalent en

#+ hexadécimal de Control-M.

echo >&2 # Le '-s' rend la frappe invisible, donc il est nécessaire d'aller

#+ explicitement sur la nouvelle ligne.

read -n 1 -s -p $'Control-J place le curseur sur la ligne suivante. \x0a'

# '0a' est l'équivalent hexadécimal de Control-J, le retour chariot.

echo >&2

###

read -n 1 -s -p $'Et Control-K\x0bva en bas.'

echo >&2 # Control-K est la tabulation verticale.

# Un meilleur exemple de l'effet d'une tabulation verticale est :

var=$'\x0aCeci est la ligne du bas\x0bCeci est la ligne du haut\x0a'

echo "$var"

# Ceci fonctionne de la même façon que l'exemple ci-dessus. Néanmoins :

echo "$var" | col

# Ceci fait que la fin de ligne droite est plus haute que la gauche.

# Ceci explique pourquoi nous avons commencé et terminé avec un retour chariot,

#+ pour éviter un écran déséquilibré.

# Comme l'explique Lee Maschmeyer :

# --------------------------

# Dans le [premier exemple de tabulation verticale]... la tabulation verticale

#+ fait que l'affichage va simplement en-dessous sans retour chariot.

# Ceci est vrai seulement sur les périphériques, comme la console Linux, qui ne

#+ peuvent pas aller "en arrière".

# Le vrai but de VT est d'aller directement en haut, et non pas en bas.

# Cela peut être utilisé sur une imprimante.

# L'utilitaire col peut être utilisé pour émuler le vrai comportement de VT.

exit 0

• **Ctl-N** Supprime une ligne de texte rappelée à partir du *tampon de l'historique* 6 (sur la ligne de commande).

• **Ctl-O** Lance un *retour à la ligne* (sur la ligne de commande).

• **Ctl-P** Rappelle la dernière commande à partir du *tampon historique* (sur la ligne de commande).

• **Ctrl-Q** Sort du mode pause du terminal (**XON**). Ceci réactive le stdin du terminal après qu'il ait été mis en pause.

• **Ctl-R** Recherche arrière pour le texte dans le *tampon historique* (sur la ligne de commande).

• **Ctrl-S** Pause du terminal (**XOFF**).Ceci gèle le stdin du terminal (utilisez Ctrl-Q pour en sortir).

Un saut de ligne (*newline*) est aussi une espace blanche. Ceci explique pourquoi une *ligne blanche*, consistant seulement d'un saut de ligne, est considérée comme une espace

• **Ctl-T** Inverse la position du caractère sous le curseur avec celle du caractère précédent (sur la ligne de commande).

• **Ctrl-U** Efface une ligne de l'entrée depuis le début de la ligne jusqu'à la position du curseur. Avec certains paramétrages, **Ctrl-U** efface la ligne d'entrée entière, *quelque soit la position du curseur*.

• **Ctl-V** Lors d'une saisie de texte, **Ctl-V** permet l'insertion de caractères de contrôle. Par exemple, les deux lignes suivantes sontéquivalentes :

echo -e '\x0a'

echo <Ctl-V><Ctl-J>

**Ctl-V** est utile principalement dans un éditeur de texte.

• **Ctl-W** Lors de la saisie d'un texte dans une console ou une fenêtre xterm, **Ctl-W** efface les caractères en commençant à partir du curseur et en reculant jusqu'au premier espace blanc. Avec certains paramétrages, **Ctl-W** efface vers l'arrière jusqu'au premier caractère non alphanumérique.

• **Ctl-X** Dans certains traitements de texte, *coupe* le texte surligné et le place dans le *presse-papier*.

• **Ctrl-Y** *Colle* le texte à l'endroit où il a été supprimé (avec **Ctrl-K** ou **Ctrl-U**).

• **Ctrl-Z** Met en *pause* un job en avant-plan. Opération de *substitution* dans certains traitements de texte.

Caractère **EOF** (end-of-file) dans le système de fichiers MSDOS.

**Espace blanc** : **Fonctionne comme un séparateur, séparant les commandes ou les variables.** Les espaces blancs peuvent être des*espaces*, des *tabulations*, des *lignes blanches* ou d'une combinaison de ceux-ci. Dans certains contextes, tels que les affectations de variable, les espaces blancs ne sont pas permis et sont considérés comme une erreur de syntaxe. Les lignes blanches n'ont aucun effet sur l'action d'un script et sont donc utiles pour séparer visuellement les différentes parties. La variable $IFS est une variable spéciale définissant pour certaines commandes le séparateur des champs en entrée. Elle a pour valeur par défaut une espace blanche.

# Pour conserver les espaces blancs dans une chaîne ou dans une variable, utilisez des guillemets.

**Les guillemets et les apostrophes.**

Encadrer une chaîne de caractères avec des apostrophes a pour effet de protéger les caractères spéciaux et d'empêcher leur réinterprétation ou expansion par le shell ou par un script shell. Un caractère est « spécial » si son interprétation a une autre signification que la chaîne elle-même, comme par exemple le caractère joker \*.

bash$ **ls -l [Vv]\***

-rw-rw-r-- 1 bozo bozo 324 Apr 2 15:05 VIEWDATA.BAT

-rw-rw-r-- 1 bozo bozo 507 May 4 14:25 vartrace.sh

-rw-rw-r-- 1 bozo bozo 539 Apr 14 17:11 viewdata.sh

bash$ **ls -l '[Vv]\*'**

ls: [Vv]\*: No such file or directory

Dans le langage de tous les jours, lorsque nous mettons une phrase « entre guillemets », nous la plaçons à part et nous lui donnons une signification spéciale. Dans un script Bash, quand nous écrivons une phrase *entre guillemets*, nous la plaçons à part et nous protégeons sa signification *littérale*. Certains programmes et utilitaires réinterprètent ou étendent les caractères spéciaux placés dans une chaîne de caractères entre apostrophes. Une fonction importante des apostrophes est donc de protéger du shell les paramètres dans une ligne de commande,

tout en laissant au programme appelé la possibilité de les étendre et réinterpréter.

bash$ **grep '[Pp]remière' \*.txt**

fichier1.txt:C'est la première ligne de fichier1.txt.

fichier2.txt:C'est la Première ligne de fichier2.txt.

Notez que la version sans apostrophes **grep [Pp]remière \*.txt** fonctionne avec le shell Bash. 1

Les guillemets peuvent supprimer l'appétit d'echo pour les nouvelles lignes.

bash$ **echo $(ls -l)**

total 8 -rw-rw-r-- 1 bo bo 13 Aug 21 12:57 t.sh -rw-rw-r-- 1 bo bo 78 Aug 21 12:57 u.sh

bash$ **echo "$(ls -l)"**

total 8

-rw-rw-r-- 1 bo bo 13 Aug 21 12:57 t.sh

-rw-rw-r-- 1 bo bo 78 Aug 21 12:57 u.sh

**Les caractères d’échappement.**

**\n** passe à la ligne

**\r** renvoie le curseur en début de ligne

**\t** tabulation

**\v** tabulation verticale

**\b** retour en arrière

**\a** « alerte » (sonore ou visuelle)

**\0xx** Transcode en octal le caractère dont le code ASCII est *0xx*

**\"** donne au guillemet sa signification littérale

echo "Bonjour" # Bonjour

echo "\"Bonjour\", a-t-il dit." # "Bonjour", a-t-il dit.

**\$** donne au dollar sa signification littérale (un nom de variable suivant un \$ ne sera pas référencé)

echo "\$variable01" # donne $variable01

**\\** donne à l'antislash sa signification littérale

echo "\\" # donne \

# Alors que...

echo "\" # Appelle une deuxième invite de la ligne de commande.

# Dans un script, donne un message d'erreur.

Le comportement de **\** est dicté par son « auto-échappement », sa mise entre guillemets ou son apparition dans une substitution de commandes ou dans un document en ligne.

# Simple échappement et mise entre guillemets

echo \z # z

echo \\z # \z

echo '\z' # \z

echo '\\z' # \\z

echo "\z" # \z

echo "\\z" # \z

# Substitution de commandes

echo `echo \z` # z

echo `echo \\z` # z

echo `echo \\\z` # \z

echo `echo \\\\z` # \z

echo `echo \\\\\\z` # \z

echo `echo \\\\\\\z` # \\z

echo `echo "\z"` # \z

echo `echo "\\z"` # \z

# Document en ligne

cat <<EOF

\z

EOF # \z

cat <<EOF

\\z

EOF # \z

.

Les éléments d'une chaîne de caractères affectée à une variable peuvent être échappés, mais le caractère

d'échappement seul ne devrait pas être affecté à une variable.

variable=\

echo "$variable"

# Ne fonctionne pas et donne le message d'erreur :

# test.sh: : command not found

# Un échappement seul ne peut être affecté correctement à une variable.

##

Ce qui arrive ici est que "\" échappe le saut de ligne et

#+ l'effet est variable=echo "$variable"

#+ affectation invalide de variable

variable=\

23skidoo

echo "$variable" # 23skidoo

# Ça fonctionne car la deuxième ligne est une affectation

#+ valide de variable.

variable=\

# \^ échappement suivi d'une espace

echo "$variable" # espace

variable=\\

echo "$variable" # \

variable=\\\

echo "$variable"

# Ne fonctionnera pas et donne le message d'erreur :

# test.sh: \: command not found

##

La première séquence d'échappement échappe la deuxième, mais la troisième

est laissée

#+ seule avec le même résultat que dans le premier exemple ci-dessus.

variable=\\\\

echo "$variable" # \\

# Deuxième et quatrième séquences d'échappement

# Ça marche.

Échapper une espace peut empêcher la séparation de mots dans une liste d'arguments pour une commande.

liste\_fichiers="/bin/cat /bin/gzip /bin/more /usr/bin/less /usr/bin/emacs-20.7"

# Liste de fichiers comme argument(s) d'une commande.

# On demande de tout lister, avec deux fichiers en plus.

ls -l /usr/X11R6/bin/xsetroot /sbin/dump $file\_list

echo "-------------------------------------------------------------------------"

# Qu'arrive-t'il si nous échappons un ensemble d'espaces ?

ls -l /usr/X11R6/bin/xsetroot\ /sbin/dump\ $file\_list

# Erreur: les trois premiers fichiers sont concaténés en un seul argument pour 'ls -l'

# parce que les deux espaces échappés empêchent la séparation des arguments

(mots).

L'échappement permet également d'écrire une commande sur plusieurs lignes. Normalement, chaque ligne séparée constitue une commande différente mais un échappement à la fin d'une ligne *échappe le caractère de saut de ligne*, et la séquence de la commande continue sur la ligne suivante.

(cd /source/repertoire && tar cf - . ) | \

(cd /dest/repertoire && tar xpvf -)

# Répétant la commande de copie de répertoires d'Alan Cox, mais séparée en deux lignes

# pour accroître la lisibilité.

# Comme alternative :

tar cf - -C /source/directory . |

tar xpvf - -C /dest/directory

# Voir note ci-dessous.

**Note**

Si la ligne d'un script termine avec un |, le caractère tube, alors il n'est pas strictement nécessaire de mettre un

échappement \. Il est néanmoins considéré comme une bonne pratique de programmation de toujours échapper une

ligne de code qui continue sur la ligne suivante.

echo "foo

bar"

#foo

#bar

echo

echo 'foo

bar' # Pas encore de différence.

#foo

#bar

echo

echo foo\

bar # Saut de ligne échappé.

#foobar

echo

echo "foo\

bar" # Pareil ici, car \ toujours interpreté comme un échappement à l'intérieur de

# guillemets faibles.

#foobar

echo

echo 'foo\

bar' # Le caractère d'échappement \ est pris littéralement à cause des guillemets

forts.

#foo\

#bar

**Exemple 5.2. Caractères d'échappement**

#!/bin/bash

# escaped.sh: caractères d'échappement

echo; echo

# Échapper un retour à la ligne.

# ------------------------------ Guillemets et apostrophes

echo ""

echo "Ceci s'affichera sur deux lignes."

# Ceci s'affichera

# sur deux lignes.

echo "Ceci s'affichera \

sur une lignes."

# Ceci s'affichera sur une ligne.

echo; echo

echo "============="

echo "\v\v\v\v" # Affiche \v\v\v\v litérallement.

# Utilisez l'option -e avec 'echo' pour afficher les caractères d'échappement.

echo "======================"

echo "TABULATIONS VERTICALES"

echo -e "\v\v\v\v" # Affiche 4 tabulations verticales.

echo "======================="

echo "GUILLEMET DOUBLE"

echo -e "\042" # Affiche " (guillemet, caractère octal ASCII 42).

echo "======================="

# La construction $'\X' rend l'option -e inutile.

echo; echo "RETOUR CHARIOT ET SON"

echo $'\n' # Retour chariot.

echo $'\a' # Alerte (son).

echo "======================="

echo "GUILLEMETS"

# Les version 2 et ultérieures de Bash permettent l'utilisation de la

#+ construction $'\nnn'.

# Notez que, dans ce cas, '\nnn' est une valeur octale.

echo $'\t \042 \t' # Guillemet (") entouré par des tabulations.

# Cela fonctionne aussi avec des valeurs hexadécimales, dans une construction

#+ du type $'\xhhh'.

echo $'\t \x22 \t' # Guillemet (") entouré par des tabulations.

# Merci, Greg Keraunen, pour nous l'avoir indiqué.

# Les versions précédentes de Bash permettent '\x022'.

echo "======================="

echo

# Affecter des caractères ASCII à une var

guillemet=$'\042' # " affecté à une variable.

echo "$guillemet Ceci est un chaîne de caractères mise entre guillemets, $guillemet

" \

"et ceci reste en dehors des guillemets."

echo

# Concaténer des caractères ASCII dans une variable.

trois\_soulignes=$'\137\137\137' # 137 est le code octal ASCII pour '\_'.

echo "$trois\_soulignes SOULIGNE $trois\_soulignes"

echo

ABC=$'\101\102\103\010' # 101, 102, 103 sont les codes octals de A, B, C.

echo $ABC

echo; echo

echappe=$'\033' # 033 est le code octal pour l'échappement.

echo "\"echappe\" s'affiche comme $echappe"

# pas de sortie visible.

echo; echo

exit 0

**SHELL**

**LES VARIABLES**

1. **LES VARIABLES**

**3\_1) Les variables d’environnement.**

Pour lister l’ensemble des variables d’environnement, on utilise la commande unix set

**USER** Le nom de l'utilisateur connecté (utilisé par des programmes de style BSD).

**LOGNAME** Le nom de l'utilisateur connecté (dans les programmes de style System V).

**HOME** Le répertoire de login d'un utilisateur, mis en place par login depuis le fichier des mots de passe passwd.

**LANG** Le nom du lieu à utiliser pour les configurations locales, s'il n'est pas écrasé par LC\_ALL ou d'autres variables comme LC\_COLLATE, LC\_CTYPE, LC\_MESSAGES, LC\_MONETARY, LC\_NUMERIC, LC\_TIME.

**PATH** Une liste de préfixes de répertoires, que sh et de nombreux autres programmes utilisent pour trouver un fichier dont le chemin est incomplet. Les préfixes sont séparés par des `:'. (De même il existe **CDPATH** utilisé par certains shells pour connaître le répertoire destination d'une commande cd, **MANPATH** utilisé par man pour trouver les pages de manuel, etc...) Si une commande à exécuter n´est présente dans aucun chemin recensé par cette variable, le système ne l'exécutera pas (à moins de spécifier le chemin complet dans la ligne de commande : /bin/ls).

**PWD** Le répertoire de travail actuel, renseigné par certains shells.

**SHELL** Le nom de fichier du shell à la connexion de l'utilisateur.

**TERM** Le type de terminal utilisé pour les affichages.

**PAGER** L'application préférée de l'utilisateur pour afficher des textes (comme less ou more).

**EDITOR** L'application préférée de l'utilisateur pour éditer des textes. (telle que emacs, joe, ou vi).

**BROWSER** L'utilitaire préféré pour examiner des URLs. Une suite de noms de navigateurs séparés par des deux-points. .

**ROOTPATH** Cette variable a la même fonction que PATH, mais celle-ci liste les répertoires qui doivent être parcourus lorsque l'utilisateur [root](l%20) entre une commande.

**LDPATH** Cette variable contient une liste de répertoires séparés par des deux-points dans lesquels l'éditeur de liens dynamiques cherche les bibliothèques.

**MANPATH** Cette variable contient une liste de répertoires séparés par des deux-points dans lesquels la commande man cherche les pages de manuel.

**INFODIR** Cette variable contient une liste de répertoires séparés par des deux-points dans lesquels la commande info cherche les pages info.

**KDEDIRS** Cette variable contient une liste de répertoires séparés par des deux-points qui contiennent les éléments spécifiques à KDE.

**CLASSPATH** Cette variable contient une liste de répertoires séparés par des deux-points qui contiennent les classes Java.

**PS1** Variable contenant une chaîne de texte réalisant la mise en forme du message d´attente (prompt) en ligne de commandes.

**PS2** Prompt de second niveau (utilisé pour des commandes tapées sur plusieurs lignes). Par défaut **>**.

**DISPLAY** Spécifie le serveur sur lequel sont lancées les fenêtres graphiques

**IFS** Liste de caractères marquant la séparation entre deux mots. Par défaut, cette variable contient les caractères << espace >>, << tabulation >> et << fin de ligne >>.

**MAIL** Contient le nom du fichier boite aux lettres de l'utilisateur.

**MAILCHECK** Temps (en seconde) entre deux vérifications de l'état de la boite aux lettres.

**MAILPATH** Liste des boites aux lettres séparées par des deux points. Par défaut le shell affiche, << **You have new mail** >> lors de la réception de nouveaux messages.

**3\_2 )** **Variables automatiquement gérées par le shell** :

**$?** valeur de retour de la dernière commande,

**$!** numéro du dernier processus ( PID ) lancé en arrière- plan,

**$$** numéro de processus ( PID ) en cours d’ éxécution,

**$#** nombre de paramètres transmis,

**$\*** ou **$@**  liste des paramètres,

**$0** nom de la commande,

$1 premier paramètre de la commande,

**$n** n-ième paramètre de la commande .

**3-3) Passage de paramètre.**

On peut récupérer facilement les compléments de commandes passés sous forme d'arguments sur la ligne de commande, à la suite du nom du script, et les utiliser pour effectuer des traitements. Ce sont les variables systèmes spéciales **$1**, **$2** …**$9** appelées paramètres de position. Celles-ci prennent au moment de l'appel du script, les valeurs des chaînes passées à la suite du nom du script (le séparateur de mot est l'espace, donc utiliser si nécessaire de " ").

Le nombre d'argument est connu avec la variable **$#**

La liste des valeurs des paramètres (même au delà des 9 premiers) s'obtient avec **$\***

**3-4) La commande shift** :

Il n'y a que 9 paramètres de position **$1** à **$9**, et s'il y a davantage de paramètres transmis, il existe une commande **shift** qui effectue un décalage de pas +1 dans les variables **$** : **$1** prend la valeur **$2**, etc

**Exemple** :

Création d'un fichier "param"

echo "Voici la liste de mes paramètres $\*"

echo $1 $2 $3 $4 $5 $6 $7 $8 $9

shift

echo $1 $9

Si je lance param de la manière suivante :

./param 1 2 3 4 5 6 7 8 9 10

J'aurais comme résultat

Voici la liste de mes paramètres 1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9

2 10

**3–5) Les variables utilisateurs**

- Le nom d une variable commence par une lettre ou \_ , suivi de lettres ou chiffres.

- La création de variables se fait en leur assignant une valeur.

variable = valeur

variable = `commande` (substitue la commande par son résultat )

variable = $(< file ) ou variable = `cat file` (affectation du contenu d’ un fichier )

Exemple : x = 1

rep = pwd

varl = hello ( " " si plusieurs chaînes de caractères )

ou par la commande « **typeset** ».

La commande **typeset** permet de typer une variable et/ou de lui affecter une valeur

**typeset** [ -fiilrux ] [ variable [= valeur]]…

-**f** : variable est une fonction,

-**i** : variable est de type entier,

-**l** : majuscules transformées en minuscules,

-**r** : variable accessible qu’ en lecture,

-**u** : minuscules transformés en majuscules

-**x** : variable exportée.

Les mêmes options avec le caractère + à la place du caractère – enlèvent l’attribut correspondant à la variable.

Exemple :

**typeset –r** v=abc définition d une variable v impossible

**echo $v**

*abc*

**typeset** v=quer

/bin/ksh : v is read only modification de v impossible

w=3+5

**echo** $w

3+5

l’expression n’est pas évaluée

**typeset –i** w demande d interprétation de w comme

**echo** $w entier

8

**typeset +r** v déportation de v

v=123

**echo** $v

123

* L’accès à la valeur de la variable se fait en utilisant "**$**" devant le nom de la variable. Les caractères  "**{}**"   permettent d’isoler le nom d’ une variable au sein d’ une chaîne.

Exemple : **echo** la valeur de x est $x

la valeur de x est 1

echo $rep

/…/…/

echo ${var1}world

*helloworld*

Les commandes **set** et **env**, utilisées sans argument donnent respectivement la liste des variables locales ( définies dans le processus **shell** actuel ) et la liste des variables d’ environnement.

**3-3 ) Le mécanisme de substitution**

**a ) ${#variable} :** longueur de la variable

## Exemple

X=abc

**print ${#X}**

*3*

**b) ${variable:-mot}:**

Valeur de variable si elle est définie et non nulle, sinon valeur de mot. Cette substitution est utilisée pour fournir une valeur par défaut quand une variable n’est pas définie.

Exemple :

X=abc

Print ${X:-cde}

abc

unset X

print ${X:-cde}

cde

**c) ${variable:=mot**} : valeur de variable, si elle est définie et non nulle, sinon la variable prend la valeur mot.

Exemple :

VAR=abc

print ${VAR:=def}

abc

unset VAR

print ${VAR:=def}

def

print $VAR

def

**d) ${variable:?mot}** :

Valeur de variable si elle est définie et non nulle , sinon valeur de mot et sortie. Si le mot est omis " *paraméter null or not set* " est affiché (souvent utilisé dans les scripts **shell** pour vérifier que les variables obligatoires sont positionnées)

Exemple :

unset VAR

print ${VAR:?variable vide}

/bin/ksh: VAR: variable vide

**e) ${variable:+mot}**

Valeur de mot si variable est définie et non nulle sinon pas de substitution (opposé de **$ { variable : -mot }**).

Exemple : VAR=abc

print ${Y:+def}

def

unset VAR

print ${Y:+def}

résultat vide

**f) ${variable#pattern} (resp. ${variable##pattern}:**

Valeur de variable à laquelle on extrait, en partant de la gauche , la plus petite partie (resp. la plus grande) correspondant au *pattern* (manipulation de sous- chaînes de caractères).

Exemple :

X=123456789

print ${X#123\*}

456789

print ${X##\*456}

789

**g) ${variable%pattern} (resp. ${variable%%pattern})**

idem **#** et **##** mais en partant de la droite.

Exemple :

X=source.c

print ${X%.\*}

source

**3-4) Tableau de variables**

En assignant une valeur X[1], la variable X est transformée en tableau de variables

Exemple :

X=A

X[1]=B

N.B: la première affectation n'est pas perdue (X[0]=A).

1. **affectations et déclarations**.

Variable[0]=valeur variable[1]=valeur …. Variable[n]=valeur

Ou

Set –A variable valeur0 valeur1 valeur2 valeur3

Ou

typeset variable[0]=valeur variable[1]=valeur …. Variable[n]=valeur

Exemple set –A Jour Lundi Mardi Mercredi Jeudi Vendredi Samedi

print ${Jour[1]} résultat : Mardi

1. **Accès**

* à un élément : print ${Jour[3]}

Jeudi

* à tous les éléments (\* ou @ ) : print ${Jour[\*] ou print ${jour[@]}
* au nombre d'élément du tableau (#) :print ${#Jour[\*]} (ou print ${#Jour[@]})

6

* à partir d'expressions arithmétiques ou d'expansion de variables :

Exemple 1 : print ${Jour[4/2]}

Mercredi

Exemple 2 : X=5

Print ${Jour[$X]}

Samedi

**3-5) Portée des variables**

Chaque processus créé par le **shell** s'exécute en héritant d'un environnement par défaut. Celui-ci comprend certaines variables définies lors de la connexion (exemple : **PATH HOME** etc…)

Les variables d'environnement sont transmises à tous les processus fils du shell et donc en particulier par le shell de login à ses processus fils.

Les variables définies dans un processus sont locales à ce processus. La commande **export** permet de rendre accessibles des variables dans un **sous-shell (processus fils).**

Il n'existe aucun moyen d'exporter une variable d'un sous-shell vers son père

**Exercice1** :

Soit le script "Processus1" Soit le script "Processus2"

VAR=123 VAR=456

echo $VAR echo $VAR

echo Numéro de processus1 echo Numéro de processus2

./Processus2

echo $VAR

Lancer le processus "Processus1". Quelles sont les Valeurs de VAR et le PID

### Exercice 2

Soit le script "Processus1" Soit le script "Processus2"

VAR=123 VAR=456

echo $VAR echo $VAR

echo Numéro de processus1 echo Numéro de processus2

. ./Processus2

echo $VAR

Lancer le processus "Processus1". Quelles sont les Valeurs de VAR et le PID

.

#### Exemple 1 :

X=bonjour

export X

ksh création d'un second shell

echo $X

bonjour la variable est connue dans le sous-shell

X="au revoir" changement de valeur

echo $X

au revoir

CTRL-D retour au shell de départ

echo $X

bonjour valeur de X inchangée

**Exemple 2** :

VAR=Vivaldi

echo $VAR

Vivaldi

ksh création d'un second shell

echo $VAR

-

CTRL-D retour au shell initial

echo $VAR1

Vivaldi

**SHELL SCRIPTS SHELL SCRIPTS**

Un **script shell** est un fichier de type texte contenant une suite de commandes **shell**, exécutable par l'interpréteur (ex *:/bin/bash,/bin/ksh*), comme une commande unique. Un script peut être lancé en ligne de commande, comme dans un autre script.

On peut définir des variables et utiliser des structures de contrôle, ce qui lui confère le statut de langage de programmation interprété et complet.

Le langage shell gère notamment :

* la gestion des entrées-sortie et leur redirection
* des variables définies par le programmeur et des variables systèmes
* le passage des paramètres
* des structures conditionnelles et itératives
* des fonctions internes.

**1) Saisie du script** :

Utiliser un éditeur de texte (**vi** par exemple)

Les lignes commençant par "**#**"sont des commentaires

Le script doit débuter par l'indication de son interpréteur écrite sur la première ligne :

**#!bin/ksh**

**Exemple** :

#!bin/ksh

# script bonjour

# affiche un salut qui l'a lancé

# la variable $USER contient le nom de login

echo ------------Bonjour $USER ---------------

# l'option –n empêche le passage à la ligne suivante

echo –n "Nous sommes le "; date

#recherche de $USER en début de ligne dans un fichier passwd

# puis l'extraction de l'uid au 3 ème champ, et affichage

echo "Ton numéro d'utilisateur est " $[grrep "^USER" /etc/passwd | cut –d:

**2) Exécution du script.**

- Pour exécuter un script, il doit posséder la permission "**x"** (soit exécutable). On peut changer cette permission pour tous les utilisateurs avec **chmod**

**chmod a+x** bonjour

Ou bien faire précéder le nom du shell avant le nom du script ex : /bin/ksh ./bonjour

- Pour lancer l'exécution du script, taper ./bonjour, ./indiquant le répertoire courant si "**.**" n'est pas défini dans la variable **PATH**

**3) Mise au point, débogage.**

Exécution en mode "trace" (**-x**) et en mode "verbeux" (**-v**)

/bin/sh –x ./bonjour

Pour aider à la mise au point d'un script, on peut insérer des lignes temporaires **: echo $**var pour afficher le contenu de la variable par exemple**, exit 1** pour forcer l'arrêt du script.

**4 ) Passage de paramètres**.

On peut passer des arguments à la suite du nom du script, séparés par des espaces. Les valeurs de ces paramètres sont récupérables dans le script grâce aux paramètres de position **$1,$2**,… mais contrairement aux langages de programmation classique, ils ne peuvent pas être modifiés.

Exemple :

#!/bin/ksh

# appel du script ./bonjour nom prenom

if [ $# = 2 ]

then

echo "Bonjour $2 $1 et bonne journée ! "

else

echo "Syntaxe : $0 nom prenom"

fi

**5) Entrée-sortie**

Ce sont les voies de communications entre le programme et la console :

* **echo**, affiche son argument texte entre guillemets sur la sortie standard, c'est-à-dire l'écran. La validation d'une commande **echo** provoque un saut de ligne

exemple : **echo** "Bonjour à tous !"

On peut insérer les caractères spéciaux habituels, qui seront interprétés seulement si l'option **–e** suit **echo**

Caractères spéciaux : **\n** (saut ligne), **\b** (retour arrière), **\t** (tabulation), **\a** (alarme), **\c** (fin sans saut de ligne)

echo "bonjour \nà tous !"

echo –e "bonjour \nà tous"

echo -e "bonjour \nà tous \net à tous ! \c"

* **read**, permet l'affectation directe par lecture de la valeur, saisie sur l'entrée standard au clavier

**read** var1 var2 … attend la saisie au clavier d'une liste de valeurs pour les affecter, après validation globale, respectivement aux variables var1 var2

Exemple:

echo "Donnez votre prenom et votre nom"

read prenom nom

echo "bonjour $prenom $nom"

**6 )** **la commande shift**

Il n'y a que 9 paramètres de position **$1** à **$9**, et s'il y a davantage de paramètres transmis, la commande **shift** effectue un décalage de pas +1 dans les variables **$** : $1 prend la valeur de $2, etc…

**7 )** **la commande set**

initialise les variables $1 $2 $3 etc…

a=1; b=2; c=3

set a b c

echo $1 $2 $3

# les valeurs de a, b, c sont récupérées dans $1, $2, $3

date

Mon March 14

set `date`

echo $1 $2 $3

$1=Mon $2=March $3=14

**8 )** **La commande test**

L'outil **test** est un outil qui permet de tester les attributs d'un fichier, de comparer des chaînes de caractères et des valeurs numériques.

Syntaxe : **test** exp ou [ exp ]

test évalue l'expression exp.

Si la valeur varie alors le code retour de test est "**0**" (vrai) sinon le code est différent de "**0**" (faux).

L'expression exp peut être construite à partir des options suivantes :

**-r** f1 vrai si le fichier f1 existe et est lisible

**-w** f1 vrai si le fichier existe et est modifiable

**-x** f1 vrai si le fichier existe et est exécutable

**-f** f1 vrai si le fichier existe et est un fichier ordinaire

**-d** f1 vrai si le fichier existe et est un répertoire

**-s** f1 vrai si le fichier existe et sa taille > 0

**-c** f1 vrai si le fichier existe et est de type caractère

**-b** f1 vrai si le fichier existe et est de type bloc

**-p** f1 vrai si le fichier existe et est de type tube nommé

**-u** f1 vrai si le fichier existe et est un fichier qui possède le bit SUID

**-g** f1 vrai si le fichier existe et est un fichier qui possède le bit SGID

**-z** var vrai si la chaine contenu dans var est de longueur nulle

-n var vrai si la chaine contenu dans var est de longueur non nulle

var1 **=** var2 vrai si les chaines contenues dans var1 et var2 sont identiques

var1 **!=** var2 vrai si les chaines contenues dans var1 et var2 sont différentes

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

n1 **–ne** n2 vrai si les entiers n1 et n2 sont inégaux

n1 **–gt** n2 vrai si les entiers n1 > n2

n1 **–lt** n2 vrai si les entiers n1 < n2

n1 **–le** n2 vrai si les entiers n1 <= n2

n1 **–ge** n2 vrai si les entiers n1 >=n2

En outre, on peut combiner ces différentes options avec les opérateurs

**!** négation

**-a** ET logique

**-o** OU logique

On peut également utiliser les parenthèses pour grouper des expressions. Il faut alors les faire précéder du caractères d'annulation " **\** " pour qu'elles ne soient pas interprétées par le **shell**

Ex: [ expr1 –a expr2 ] (ET) vrai si les 2 expressions sont vraies

[ expr1 -o expr2 ] (OU) vrai si l'une des deux expression est vrai

[ ! expr1 ] négation

**9 ) Les structures de contrôle**.

**9-1 ) Structures conditionnelles**.

**if** suite-de-commandes

**then**

# séquence exécutée si "suite-de-commandes" rend une valeur 0

bloc-instruction1

**else**

# séquence exécutée sinon

bloc-instruction2

**fi**

Attention! si **then** est placé sur la même ligne que le **if**, séparer avec "**;"**

if commande; then

…

**Exemples** :

1. toto possède t-il un compte? on teste la présence d'une ligne commençant par toto dans /etc/passwd ( >/dev/null pour détourner l'affichage de la ligne)

**if** grep "^toto" /etc/passwd > /dev/null

**then**

echo "toto a deja un compte"

**fi**

2- Si toto a une bonne note, on le félicite

note=17

**if** [ $note **–gt** 16 ] ---- test vrai, valeur retournée : 0

**then** echo " tres bien!"

**fi**

3- Avant d'exécuter un script, tester son existence.

**if** [ -f ~/.bashrc ]

then

.~/.bashrc

**fi**

**9-2 ) Conditionnelles imbriquées.**

pour imbriquer plusieurs conditions, on utilise la construction :

**if** commande1

then

bloc-instruction1

**elif** commande2

then

bloc-instruction2

**else**

# si toutes les conditions précédentes sont fausses

bloc-instruction3

**fi**

**Exemple** :

1 – toto a t-il fait son devoir lisiblement ?

fichier=/home/tot/devoir.doc

**if** [ -f $fichier **–a –r** $fichier ]

**then**

echo "je vais vérifier ton devoir"

elif [ **! –e** $fichier ]

**then**

echo "ton devoir n'existe pas"

2 – Supposons que le script exige la présence d'au moins un paramètre, il faut tester la valeur de **$#**, est-elle nulle ?

**if** [ $# = 0 ]

then

echo "erreur, la commande exige au moins un argument .."

exit 1

**elif** [ $# = 1 ]

**then**

echo "Donner la second argument : "

read arg2

**fi**

**9-3 ) la structure case**

syntaxe: **case** var **in**

cas1) commande1 ;;

cas2) commande2 ;;

casn) commanden ;;

**esac**

Exemples :

1 – supposons que le script doive réagir différemment selon l'utilisateur courant; on va faire plusieurs cas selon la valeur de $USER

**case** $USER **in**

root) echo "Bonjour Mr l'administrateur $USER" ;;

jean | etu?) echo "Bonjour Mr l'etudiant $USER " ;;

toto ) echo " Attention ne fais pas le zigoto \! " ;;

**esac**

2 – Le script attend une réponse oui/non de l'utilisateur

read reponse

**case** $reponse **in**

[oOyY]\*) ……………. ;;

[nN]\*) ……………….. ;;

**esac**

3 – **case** $param **in**

0|1|2|3|4|5|6|7|8|9 ) echo $param est un chiffre ;;

[0-9]\*) echo $param est un nombre ;;

[a-zA-Z]\*) echo $param est un nom ;;

\*) echo $param de type non prevu ;;

**esac**

4 – un exemple, extrait du script smb (/etc/rc.d/init.d/smb)

**case** "$1" **in**

start)

echo –n "Starting SMB services : "

deamon smbd –D

echo –n "Starting NMB services : "

deamon nmbd –D

… ;;

stop)

echo –n "Shutting SMB services : "

killproc smbd

… ;;

**esac**

**9-4) structures itératives**.

1. **Boucle for**

Syntaxe :

**for** variable [ in liste ]

**do**

commandes (utilisant $variable)

**done**

* **fonctionnement**.

Ce n'est pas une boucle **for** contrôlée habituelle fonctionnant comme dans les langages de programmation classiques (utiliser pour cela une boucle **while** avec une variable numérique)

La variable parcours un ensemble de fichiers donnés par une liste ou bien implicitement et le bloc commandes est exécuté pour chacune de ses valeurs.

Les mots clés **do** et **done** apparaissent en début de ligne (ou après un ";")

Exemples:

la liste peut être explicite

**for** nom **in** jean toto etu1

**do**

echo "$nom, à bientot"

**done**

la liste peut être calculée à partir d'une expression modèle

#recopier les fichiers personnels de toto dans /tmp/toto

**for** fich **in** /home/toto/\*

**do**

cp $fich /tmp/toto

done

Si aucune liste est précisée, les valeurs sont prises dans la variable système $@, c'est-à-dire en parcourant la liste des paramètres positionnels courants.

# pour construire une liste de fichiers dans $@

**cd** /home/etu1 ; **set** \* ; **echo** $@

**for** nom **in** $@

**do**

echo $nom

**done**

Expliquez les exemples suivants :

1 - ) **for** nom in /home/stage[1-9]

**do**

**echo** "$nom, tout va bien ?"

**done**

2 - ) **for** i **in** /home/\*/\* ; **do** echo $i ; **done**

3 -) **for** x **in** /home/st\*

**do**

echo $x >> liste-re-stage.txt

**done**

4 -) **for** x **in** $(grep "^st" /etc/passwd | cut –d: -f6)

**do**

echo $x; echo $x >> $HOME/tmp/liste-rep-stage.txt

**done**

more liste-rep-stage.txt

**b )la boucle while**.

**while** liste-commandes ) la répétition se poursuit TANT QUE la dernière

**do** ) commande de la liste est vrai (c-à-d renvoie un code

commandes ) retour nul)

**done**

**until** liste-commandes ) la répétition se poursuit JUSQU'A CE QUE la dernière

**do** ) commande de la liste devienne vraie

commandes

**done**

##### Voici 2 exemples à comparer

**echo** –e "Entrez un nom de fichier"

**read** fich

**while** [ -z "$fich" ]

**do**

**echo** –e "saisie à recommencer"

**read** fich

**done**

**while**

**echo** –e "entrez un nom de fichier"

**read** fich

[ -z "$fich" ]

**do**

**echo** –e "Saisie à recommencer"

**done**

**Exemples à tester** :

# pour dire bonjour toutes les 1à secondes (arrêt par CTR-C)

**while** true ;

**do**

**echo** 'Bonjour Mr $USER"

**sleep** 10

**done**

Lecture des lignes d'un fichier pour traitement : noter que la redirection de l'entrée de la commande **while … do … done** est placée à la fin

fich=/etc/passwd

**while** read ligne

**do**

**echo** $ligne

…..

**done <$**fich

# c) Sortie et reprise de boucle

* **break** placé dans le corps d'une boucle, provoque une sortie définitive de cette boucle.
* **continue** permet de sauter les instructions du corps de la boucle (qui suivent continue) et de "continuer" à l'itération suivante.

Pour les boucles **for**, **while** et **until**, continue provoque donc le réévaluation immédiate du test de la boucle.

Exemples importants :

Boucle de lecture au clavier arrêtée par la saisie de stop

#!/bin/bash

# syntaxe : lecture.sh

texte=" "

**while** true

**do**

read ligne

**if** [ $ligne = stop ]

**then** break

**else** texte="$texte \n$ligne"

**fi**

**done**

**echo** –e $texte

Lecture des lignes d'un fichier.

fich="/etc/passwd"

**grep** "^stage" $fich | **while** true

**do**

read ligne

**if** [ "$ligne" = " " ] ; **then** break ; **fi**

echo $ligne

**done**

**10 ) Fonctions**

2 syntaxes

**function** nom-fct {

bloc d'instructions

}

**nom-fct()** {

bloc d'instructions

}

Exemple :

En connexion **root**, on doit relancer les "démons", si on a modifié un fichier de configuration. Par exemple, **/etc/rc.d/init.d/smb** contient la commande deamon smbd –D, pourtant à l'essai deamon est une commande inconnue!

Reportons nous au début du fichier, le script /etc/rc.d/init.d/functions y est appelé. Celui-ci contient la fonction: deamon() { …

- passage d'arguments : identique que les script

* variables locales : Dans le corps de la fonction, on peut définir et utiliser des variables déclarées locales, en les introduisant avec le mot-clé **local**

**11 ) Commandes diverses.**

**11-1) Calcul sur les entiers relatifs**.

Ne pas confondre la syntaxe **$((expression arithmétique))** avec la substitution de commande **$(commande)**

Les priorités sont gérées par un parenthèsage habituel

echo $((30+2\*10/4))

echo $(( (30+2) \* (10-7) /4 ))

**11-2) la commande tr**

Cette commande de filtre permet d'effectuer des remplacements de caractères dans une chaine.

Par exemple pour transformer une chaine en minuscule :

chaine= "Bonjour, coment allez VOUS aujourd'hui ?"

echo $chaine | tr 'A-z' 'a-z'

Pour permettre l'utilisation de la commande set (voir ci-dessous), il est nécessaire que le séparateur de champ sur une ligne soit l'espace, et non pas par exemple :

créer un fichier passwd.txt qui introduit un espace à la place de ":" dans une copie de /etc/passwd

cat passwd | tr ":" " " > passwd.txt

**11-3) la commande set**

Cette commande interne est très pratique pour séparer une ligne en une liste de mots, chacun de ces mots étant affecté à une variable positionnelle. Le caractère de séparation est l'espace.

# soit une chaine ch qui contient une liste de mots

c="prof eleve classe note"

# set va lire chaque mot de la liste et l'affecter aux paramètres de position

set $c ; echo $1 $2 $3 $4

shift ; echo $1 $2 $3 $4

Le langage shell est inadapté aux calculs numériques. Mais si vraiment on veut calculer (sur des entiers) …

Exemple :

declare –i k ; k=1 ; p=1

while [ $k –le 10 ]

do echo "$k != "$((p=$p \* $k)) ; k=$k+1

done

**11-4 ) la commande eval**

Cette commande ordonne l'interprétation par le **shell** de la chaine passée en argument. On peut ainsi construire une chaine que l'appel à **eval** permettra d'exécuter comme une commande.

Exemple :

message="quelle est la date d'aujourd'hui ?"

set $message

echo $# le nombre de mots est 6

echo $4 date

eval $4 interprète la chaine "date" comme une commande

Il est souvent pratique de construire une chaine dont la valeur sera égale au libellé d'un enchainement de commande (par ";"). Pour faire exécuter ces commandes contenues dans la chaine, on la passe comme argument de la commande **eval**

exemple 1

liste="date;who;pwd" ( ' ' ou " " obligatoires sinon le ; est un séparateur

eval $liste

------------ exécute bien les 3 commandes.

exemple 2

Soit la chaine $user qui contient des informations sur un compte à créer. S'il utilise un autre séparateur que ";" on fait appel à la commande **tr** d'abord

user="login=toto ; mdp=moi ; nom=toto ; groupe=profs"

eval $user

echo $login $mdp $nom $groupe

useradd –G $groupe $login

echo $mdp | (passwd --stdin $login)

**EXPRESSIONS RATIONNELLES** **Généralités sur les expressions rationnelles**

## Introduction

## Le livre de référence utilisé : Introduction à PERL (O'Reilly)

* Les expressions rationnelles (ou régulières) sont des critères ou modèles de recherche (pattern) dans les chaines de caractères. Les objectifs peuvent être simplement de sélectionner suivant ce critère, ou d'effectuer des traitements comme des subtitutions sur les chaines trouvées.
* Leur utilisation s'étend de certains filtres shell : grep, sed, awk, vi , emacs .. à des langages de scripts : perl, php .. et aux éditeurs de texte : vi , emacs
* Attention ! certains caractères spéciaux sont communs avec les caractères génériques de désignation de fichiers, mais ils ont une interprétation différente. Donc il faut toujours prêter attention au contexte

###### Quelques exemples avec grep

* grep abc fichier recherche la chaine **abc** dans toutes les lignes du fichier. Les lignes trouvées sont envoyées sur la sortie standard, éventuellement redirigée.
* grep " " fichier recherche les lignes qui contiennent un (et un seul) espace entre 2 mots
* grep "ab\*c" fichier, idem avec 0 ou plusieurs occurrences de la lettre b
* grep "^s.\*n$" fichier, reconnait les lignes débutant par s (^s), finissant par n (n$), avec éventuellement des caractères (quelconques) intermédiaires (.\*)
* grep "^[a-zA-Z][a-zA-Z0-9]\*" [a-f]\*.txt recherche les chaines commençant par une lettre suivie d'un nombre qcq de caractères alphanumériques, dans tous les fichiers dont les noms débutent par une lettre de a à f avec une extension .txt

#### Quelques exemples en PERL

* Une expression rationnelle est une chaine encadrée (en PERL) par **/../**, appelé **opérateur de correspondance**, qui applique le modèle par défaut sur la variable **$\_**, et renvoie vrai ou faux.  
  Par exemple, sur la ligne courante d'un fichier, contenue dans **$\_**, recherche de la séquence **az** :
* if (/az/) {
* print $\_ ;

}

* Pour appliquer le modèle à une autre variable, il faut utiliser l'opérateur **=~** comme dans l'exemple :
* if ($var =~ /az/) {
* print $var ;

}

* Exemples script cherche-az, pour rechercher la séquence "az":
* *# pour opérer sur toutes les lignes saisies au clavier ou récupérées sur l'entrée standard*
* *# rappel : chaque ligne de l'entrée est stockée dans l'argumement par défaut $\_*
* *# appel : cat fichier |cherche-az*
* while () {
* if (/az/) { print $\_ ; }
* }
* *# pour opérer sur toutes les lignes d'un fichier*
* *# appel cherche-az*
* open(IN, "~/essai.txt");
* while () {
* chomp($\_);
* if (/az/) { print "$\_ \n"; }
* }

## Les méta-caractères

* Appelés aussi caractères spéciaux, ce sont des caractères interprétés *en contexte expression rationnelle* comme des opérateurs.
* En voici la liste avec un bref descriptif :
* . (point) représente un caractère qcq, sauf \n
* \* (astérisque) répétition du caractère précédent
* + au moins une occurence de l'expression régulière
* ? au plus une occurence de l'expression régulière
* [...] (crochets) l'un des caractères de l'ensemble.
* [^..] en début de crochets recherche dans le complémentaire de l'ensemble
* ^ recherche en début de ligne
* $ recherche en fin de ligne
* \ annule le rôle de méta-caractère, pour jouer le rôle du caractère usuel
* {n,m} indique le nombre de répétitions attendus du caractère précédent
* | joue le rôle de "ou" entre 2 expr rég.
* L'antislash **\** inhibe l'interprétation des caractères spéciaux et force leur interprétation usuelle.  
  Exemples
* **.\.txt** recherche les chaines du genre c.txt , où c est un caractère unique qcq
* **\\*$** recherche les chaines qui se terminent ($) par le caractère astérisque (\\*)

## Ecriture des motifs

### Expression régulière à 1 caractère (atomique)

* un caractère correspond à lui-même, en règle générale.  
  Ainsi le motif **c** recherche le caractère désigné par c.
* un métacaractère précédé de \, pour lui rendre son rôle usuel.
* **.** Le point remplace tout caractère unique, sauf \n (newline)
  + **a.** toute suite de 2 caractères commençant par a, sauf **a\n**
  + **b.c** désigne toute suite de 3 caractères du genre bac, bbc, bcc, ...
* **[abc]** classe de caractères, sélectionne toute chaine contenant l'un des caractères listés
  + **[a-z]** toute lettre minuscule
  + **[0-9]** équivaut à [0123456789], un chiffre quelconque
  + **[a-zA-Z0-9\-\_]** correspond à n'importe quelle lettre ou chiffre, ou au tiret ou au souligné
* le caractère ^ juste après [ joue le rôle d'exclusion des caractères qui suivent.
  + [^aeiou] tout **sauf** une lettre voyelle
  + [^a-zA-Z0-9] sélectionne un caractère non alphanumérique
* Certaines classes sont prédéfinies et servent d'abréviations

|  |
| --- |
| **Construction** |
| **Classe équivalente** | **Construction de négation** | **Classe équivalente** |
| \d (un chiffre) | [0-9] | \D (chiffres, non !) | [^0-9] |
| \w (1 caractère de mot) | [a-zA-Z0-9] | \W (mots , non!) | [^a-zA-Z0-9] |
| \s (espace) | [ \r \n \f \t] | \S (space, non !) | [^ \r \n \f \t] |

#### Exemples

* + [\da-fA-F] correspond à un chiffre hexadécimal quelconque
  + 5[0-9][0-9] pour chercher les comptes d'UID 500 à 600.
  + a[bc]d correspond aux chaînes abd ou acd.

### Expressions régulières simples

Soit expat une *expression régulière atomique* quelconque, alors ce sont

|  |  |
| --- | --- |
| **expat1expat2expat3 ...** | toute concaténation sans espace formée d'expr. régulière atomiques |
| **expat\*** | chaine composée de 0 à N caractères vérifiant expat |
| **expat+** | chaine contenant au moins un caractère vérifiant expat |
| **expat?** | chaine contenant au plus un caractère vérifiant expat |
| **expat{n}** | chaine composée exactement de n caractères vérifiant expat |
| **expat{n,}** | chaine composée d'au moins n caractères vérifiant expat |
| **expat{n,m}** | chaine composée de n à m caractères vérifiant expat |

#### Exemples

* **a\*** caractère de multiplication, suivant un caractère signifie la répétition de 0 à plusieurs exemplaires de ce caractère.  
  **[ab]\*** signifie répétition possible d'un quelconque des 2 lettres a ou b.
* **a+** répétition de 1 à plusieurs exemplaires de a
* **a?** 0 ou 1 exemplaire de a  
  c[ad]?r impose la présence de cr ou car ou cdr, et rien de plus.
* **/x{5,10}** 5 à 10 répétitions attendues de x
* **a{5,}** 5 ou plus répétitions attendues de a
* **a{5}** exactement 5 occurrences de a
* **a.{5}b** recherche les mots contenant 5 caractères entre a et b
* \s+ recherche un ou plusieurs caractères séparateurs.
* Conséquence : {0,} {1,} {0,1} correspondent à \* , + , ?

### Expressions régulières étendues

Soit expsim une *expression régulière simple* quelconque, comme définie précédemment, alors ce sont des expressions régulières générales :

|  |  |
| --- | --- |
| **expsim1expsim2 ...** | toute concaténation formée d'expr. régulière simples |
| **^expsim** | recherche le motif expsim en **début** de chaine |
| **expsim$** | recherche le motif expsim en **fin** de chaine |
| **expsim\b** | le motif expsim doit se trouver la **fin d'un mot** |  |
| **expsim1|expsim2|...** | alternative : recherche de expsim1 ou expsim2 ... |  |
| **(expsim)\*** | chaine composée de 0 à N caractères vérifiant expsim |  |
| **(expsim)+** | chaine contenant au moins un caractère vérifiant expsim |  |
| **(expsim)?** | chaine contenant au plus un caractère vérifiant expsim |  |
| **(expsim){n}** | chaine contenant la concaténation d'exactement n chaines vérifiant expsim |  |
| **(expsim){n,m}** | chaine composée de n à m caractères vérifiant expsim |  |

#### Remarques

* Les symboles **^, $** sont appelés *motifs d'ancrage*, car ils permettent de préciser l'endroit où doit être effectuée la recherche du motif dans la chaine (alors que sans directive particulière la recherche de correspondance du motif avec la chaine s'effectue de gauche à droite de la chaine).
* Le symbole d'ancrage **\b** indique une limite de mot obligatoire, à l'endroit où il est inséré dans le motif  
  Par exemple :
* info**\b** est vérifié par "l'info pour tous", "info-matin" mais pas par infos, informatique ...

**\b**info est vérifié par "informatique","m'informer" mais pas desinformation, ..

* Les parenthèses autour de expsim sont indispensables, sinon les opérateurs de répétition s'appliquent au dernier caractère de l'expression (il s'agit d'une question de priorité des opérateurs ..)  
  Par exemple, les motifs **li(nux)\*** et **linux\*** reconnaissent respectivement li, linux, linuxnux,.. et linu, linux, linuxx, linuxxx,...
* Attention, ces extensions ne sont pas toutes reconnues par l'ensemble des filtres. Par exemple la commande grep, ne connait pas | , ni les parenthèses

#### Exemples

* color|couleur réussit si l'un des motifs, soit color, soit couleur, est trouvé
* 5[0-9]{2} reconnait tout nombre de 500 à 599.
* /^(eleve|prof|stage)[3-7]/ est satisfait par les chaines commençant par l'un des 3 mots suivis immédiatement par un numéro de 3 a 7
* comment reconnaitre un mot de 6 lettres formé des lettres (cela ne vous rappelle rien ?)

#### Parenthèses de mémorisation

La présence de parenthèses permet de mémoriser une ou plusieurs parties de l'expression qu'elle entoure, sans modifier son interprétation. Pour rappeler ces valeurs mémorisées, on utilise les notations \1, \2 ... qui correspondent aux valeurs reconnues de même ordre.  
Ainsi **eleve(\d).+poste\1** sera satisfait par *eleve3 au poste3* et non par *eleve5 au poste3*

**QUELQUES UTILITAIRES :**

**Cut, grep, tr, sed, qawk.** **Commandes de filtres : grep, cut, tr, sed**

## Généralités

* Un filtre est une commande qui lit les données sur l'entrée standard, effectue des traitements sur les lignes reçues et écrit le résultat sur la sortie standard.
* Bien sûr les entrées/sorties peuvent être redirigées, et enchainées avec des tubes.  
  A noter que le caractère d'indirection < en entrée n'est pas obligatoire pour les filtres  
  Ainsi, dans # cat /etc/\*.conf > tous.conf cat va bien lire les fichiers qui correspondent au modèle /etc/\*.conf et les contaténer dans le fichier tous.conf
* Dans ce chapitre, on va revoir ou découvrir les principaux filtres utilisés dans le monde UNIX
  + [cat, more et less](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#cat)
  + [grep](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#grep)
  + [cut](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#cut)
  + [wc](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#wc)
  + [tr](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#tr)
  + [sed](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#sed)
  + [awk](http://www.ac-creteil.fr/reseaux/systemes/linux/shell-scripts/#awk)

## 1° ) Les commandes cat, more et less

cat f1 f2 .. > f concaténe f1, f2 .. dans le nouveau fichier f

less f1 f2 .. > f concaténe les fichiers f1 f2 .. en un seul fichier f (comme cat)

less f3 >> f ajoute le contenu du fichier f3 à la suite du contenu du fichier f

## 2°) La commande grep : sélection de lignes

Cet utilitaire (*General Regular Expression Parser*, analyseur général d'expression régulière) sélectionne toutes les lignes qui satisfont une expression régulière (ou rationnelle).

#### Syntaxe

**grep [options] expreg [fichiers]**   
Cette commande recherche dans les fichiers ou sur son entrée standard des lignes de texte qui satisfont l'expression régulière expreg indiquée.  
Sa sortie peut être redirigée dans un fichier.

**options**

-c         donne seulement le nombre de lignes trouvées obéissant au critère   
-l          donne seulement le nom des fichiers où le critère a été trouvé   
-v         donne les lignes où le critère **n'a pas** été trouvé   
-i          ne pas tenir compte de la casse (ne pas différencier majuscules minuscules)   
-n          pour n'afficher que les numéros des lignes trouvées   
-w          pour imposer que le motif corresponde à un mot entier d'une ligne

**constructions**

grep est souvent inclus dans un tube qui lui fournit en entrée le fichier à étudier.

*Exemple : quelle est la question posée ?*

cat /etc/passwd | cut -d: -f1 | grep -w "jean" > sortie

#### Expressions reconnues

Grep ne reconnait pas toutes les expressions rationnelles étendues.  
Voici la liste des symboles utilisables par grep : . \* [ ] [^ ] ^ $

* **.**   signifie un caractère quelconque
* **\***   répétition du caractère situé devant
* **^**   début de ligne
* **$**  fin d'une ligne (donc "e$"  mots se terminant par e)
* **[...]**    contient une liste ou un intervalle de caractères cherchés
* **[^..]**    caractères interdits.

**Attention**

Pour éviter une confusion entre les interprétations de ces symboles spéciaux par grep ou par le shell, il est indispensable de "verrouiller" expreg en plaçant l'expression entre guillemets " " (et non entre quotes !).

#### Exemples

Etudier et commenter les commandes suivantes :

1. *cherche dans fichier, les lignes dont la 1ère lettre est qcq et la 2ème doit être o*   
   grep "^.o" fichier
2. *cherche dans le fichier passwd les lignes commençant par t*   
   grep "^t" /etc/passwd
3. *cherche les lignes ne commençant pas commençant par t*   
   grep -w "^t" /etc/passwd
4. *cherche les lignes contenant les mots suivant le modèle T.t.*   
   grep "T.t." /etc/passwd
5. *cherche dans le fichier des groupes, ceux qui commencent par a ou b .. ou j*   
   less /etc/group | grep "^[a-j]"
6. *pour lister les s-répertoires du rép. /etc*   
   ll /etc | grep "^d"
7. *compter les lignes saisies au clavier qui se termine par a*   
   grep -c "a$"
8. *afficher les lignes des fichiers essai?.txt qui contiennent a, b ou c*   
   grep [abc] "essai?.txt"
9. *détourne le flot de sortie du moniteur pour l'envoyer sur l'entrée de wc pour ?*   
   grep [abc] "essai?.txt" | wc -l

#### Exercices

1. donner une version sans cut de la commande précédente
2. *Comment ne sélectionner que "root" avec*
3. cat /etc/passwd | cut -d : -f 1 | grep "r"
4. on sait que ps aux donne la liste des processus. La commande /etc/X11/X est celle qui lance le serveur X. Il est nécessaire de connaitre son **PID**, en cas de plantage du serveur X  
   Ecrire la commande qui retourne la ligne correspondante.
5. Se placer dans /etc. En une seule commande, faire calculer et afficher le nombre de sous-répertoires de /etc, sous la forme :   
   *"Il y a 33 répertoires dans /etc"*
6. Créer un fichier essai1 contenant quelques lignes dont des lignes vides Avec **grep** générer le fichier essai2 à partir de essai1 sans ligne vide
7. cat essai1 ..... >essai2

## 3° ) cut : sélection de colonnes

La commande **cut** présente 2 formes suivant que l'on sélectionne des colonnes de caractères ou qu'on distingue des champs séparés par un caractère précis.

sélection\_colonne

**cut -c(sélection\_colonnes) [fichiers]**

**Exemples**

* *affiche le 5ième caractère*   
  cut -c5 fichier
* *affiche du 5ième* ***au*** *10ème caractères*   
  cut -c5-10 fichier
* *affiche le 5ième* ***et*** *le 10ème caractères*   
  cut -c5-10 fichier
* *affiche à partir du 5ième (jusqu'à la fin)*   
  cut -c5- fichier

**sélection champs**

**cut -d(séparateur) -f(sélection\_champs) [fichiers]**

#### Exercices

Etudier les commandes suivantes

1. cut -d":" -f1,6 /etc/group
2. *Que réalise la ligne suivante ? Vérifiez*  
   grep "^st" /etc/passwd | cut -d":" -f1,3-4,6

## 4° ) La commande wc

#### Exemples

cat /etc/paswwd | grep /bin/bash/ | wc -l

*pour compter les titulaires d'un compte pouvant se connecter avec le login shell*

less /etc/paswwd | grep -vc /bin/bash/

*négation de la question précédente (revoir les rôles ds options -c et -v)*

## 5° ) La commande tr

**tr=Translate**, est un filtre ne reconnaissant pas les expr. régulières.  
Cette commande est le plus souvent associée à des redirections  
Les caractères entrés sont traités et le résultat est envoyé sur la sortie standard   
On peut utiliser les intervalles du type a-z et les codes ASCII des caractères en notation octale \0xx

#### Syntaxe

1. **tr [options] ch1 ch2 <fich1 >fich2**   
   Remplace toutes les occurrences de TOUS les caractères de ch1 par le caractère de ch2, de même rang, dans le flot d'entrée.
2. Exemple
3. *# pour convertir et afficher la ligne saisie au clavier en minuscules*
4. read ligne; echo $ligne | tr 'A-Z' 'a-z'
5. tr **-c** chaine car remplace tout caractère NON INCLUS dans la chaine chaine par le caractère car
6. *# remplace supprime tous les caractères différents de a,b, ..z par un espace*
7. echo $ligne | tr -c a-z ' '
8. tr **-d** chaine supprime tout caractère entré, appartenant à la chaine chaine
9. *# supprime toutes les minuscules non accentuées*
10. echo $ligne | tr -d a-z
11. tr **-s** chaine supprime toute répétition des caractères contenus dans chaine
12. *# supprime les espaces multiples entre les mots*
13. echo $ligne | tr -s ' '

#### Exercices

Expliquer les effets de ces commandes :

1. tr 'a,/' 'A;\_' <fich1 >fich2
2. tr 'a-z' 'A-Z' <fich1 >fich2
3. tr -d '\011\015\032' <fich1 >fich2
4. tr -s ' \011\012' <fich1 >fich2
5. tr -cs 'a-zA-Z0-9' '\n' <fich1 | sort | uniq >fich2

## 6° ) L'utilitaire sed

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

#### Syntaxe

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

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

#### Commande de substitution

* La commande s permet d'effectuer des substitutions suivant la syntaxe :  
  **sed [adresse]s/expr-régulière/remplacement/options**
* Attention ! contrairement à ce que l'on pourrait attendre, cette commande laisse passer toutes les lignes et ne sélectionne pas celles qui ont satisfait l'expression régulière et donc subi la substitution. Pour sélectionner, voir la commande de destruction.
* Options
  + Sans précision, la commande ne s'applique qu'à la 1ère occurence de chaque ligne.
  + 0...9 : indique que la substitution ne s'applique qu'à la nième occurence
  + g : effectue les modifications sur toutes les occurences trouvées.
* Exemple : sed s/moi/toi/g fich.moi > fich.toi  
  le fichier fich.moi est parcouru, à chaque occurrence de "moi", ce mot est remplcé par "toi" et le nouveau fichier est sauvegardé sous le nom fich.toi
* Délimitateur  
  Le slash **/** étant très utilisé au niveau du shell comme séparateur de niveau de répertoire, il est possible d'utiliser à la place tout autre caractère comme **#**  
  sed s#/home#/rep\_perso#g /etc/passwd > /tmp/passwd.new

#### Destruction ou sélection

* Cette option permet de filtrer les lignes qui satisfont une expression régulière. Ces lignes ne sont pas détruites dans le fichier d'origine, mais ne sont pas transmise en sortie.
* Comment modifier alors le fichier à traiter ?
* cp fichier copie
* sed /.../d copie
* Par exemple, pour détruire toutes les lignes vide d'un fichier :

sed /^$/d

#### Ajout, insertion et modification

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

sed [adresse] commande\

expression

La commande peut être :

a pour ajout ;

i pour insertion ;

c pour modification.

Réponses

#### grep

1. less /etc/group | grep "^[a-j]"
2. ps aux | grep "/etc/X11/X"
3. cat essai1 | grep -v "^$" >essai2

#### tr

1. remplace resp. a , et / par A ; et \_
2. met en majuscules
3. suppression des caractères ASCII tab (9), retour-chariot (13) et Ctrl-Z (26)
4. suppression des répétitions des espaces, des tab et des passages à la ligne (10)
5. découpage en mots de la chaine entrée, trie des lignes et suppression des doublons

# Introduction à l'utilitaire (g)awk

#### Sources

 Linux (in a nutshell) O'Reilly éditeur (chapitre 11)

 le manuel en ligne : **man gawk**

## Fonctionnalités et origine

* Origine  
  Le nom de l'utilitaire **awk** vient de ses 3 auteurs Aho, Weinberger et Kernighan. Il peut être classé dans la catégorie des filtres. Mais c'est beaucoup plus qu'un utilitaire de gestion des fichiers textes, il intégre un langage interprété très voisin du C.   
  La version disponible sur Linux s'appelle **gawk** =GNU awk; il s'agit d'une version plus performante.
* Cet utilitaire, comme **sed**, s'applique à un fichier, ou à un flot de données provenant de son entrée, si le fichier (précédé de -f) est absent. Il lit ce fichier ligne par ligne (toutes les lignes par défaut). Les lignes sont séparées par des retour-chariots.
* Il décompose chaque ligne (enregistrement) en champs, les champs étant des chaines de caractères séparées les unes des autres par un caractère particulier appelé le **séparateur**.  
  Ce caractère de séparation peut être fourni explicitement, sous la forme de l'option -F":", par exemple, ou invoqué dans le motif d'une expression rationnelle.

## Introduction par l'exemple

Voici un mini-tutoriel qu'il est conseillé d'expérimenter.  
Il utilise le fichier /etc/passwd supposé bien connu.  
Son format de ligne est : login:x:uid:gid:commentaire:home:shell

1. Voici une syntaxe simplifiée : **gawk -F":" 'bloc-programme' /etc/passwd** 
   * L'option **-F":"** signale que le caractère séparateur est ":" dans ce fichier.
   * Le bloc-programme, entouré de **{}** s'appliquera à chaque ligne. Il contient souvent **print**, l'instruction d'affichage sur la sortie standard, et des chaines constantes entourées de " ".  
     A noter que print, seul, affiche toute la ligne et insère un retour-chariot.
   * Par exemple la commande :  
     **gawk -F":" '{print "Ligne n° " NR "=" $0}' /etc/passwd**   
     affiche les lignes (référencées par la variable système **$0**), précédées de leur numéro (contenu dans **NR**)

Comparer avec **gawk -F":" '{print}' /etc/passwd**

1. Chacune des lignes est découpée par **gawk** suivant ":" en 7 champs, dont les valeurs individuelles sont affectées aux variables système $1, $2 ... jusqu'au dernier.  
   Par exemple, soit la commande :  
   **gawk -F":" '{print "Le répertoire personnel de " $1 " est " $6}' /etc/passwd**   
   appliquée à la ligne particulière **toto:x:520:521:Monsieur Toto Jules:/home/toto:/bin/bash**, elle fera afficher :  
   *Le répertoire personnel de toto est /home/toto*
2. **gawk -F":" '{print "Le nombre de champs = " NF " et le dernier vaut " $NF }' /etc/passwd**   
   Cette commande affiche le nombre de champs pour chaque ligne et la valeur du dernier.  
   Ce nombre de champs est contenu dans la variable prédéfinie **NF**, ici 7, et la valeur du dernier champ peut alors être référencée par **$NF**
3. **gawk -F":" '{print "Premier champ : "$1 " ; print "Et dernier : $NF }' /etc/passwd**   
   Bien noter que le bloc d'instruction (appelé souvent le bloc "action") est entouré d'accolades, et que le point-virgule est le séparateur d'instructions. Dans l'exemple, à comparer au précédent, chaque print fait passer à la ligne.
4. **gawk -F":" '{ if ($3 >= 500 ) print "login " $1 " uid = " $3 " rép perso : " $6}' /etc/passwd**   
   Les utilisateurs créés par root ont un numéro, leur uid, attribué à partir de 500.  
   La commande permet de sélectionner ces utilisateurs et d'afficher leur login, leur uid et leur répertoire personnel.  
   Exercice d'application  
   Sélectionner les utilisateurs précédents, extraire leur login et leur nom complet (normalement 5ème champ $5), les trier par ordre alphabétique et les stocker dans le fichier users.txt
5. Dans l'exemple précédent, la sélection est effectuée par une expression relationnelle, autrement dit une comparaison  
   Les opérateurs de comparaison usuels peuvent s'appliquer (attention : l'égalité s'exprime avec **==**, la différence avec **!=**) ainsi que les opérateurs booléens : **&&** (et), **||** (ou) , ! (non)  
   Par exemple la commande :  
   **gawk -F":" '{ if ($3 >= 500 && substr( $1, 0, 1)== "j") print NR "login " $1 " uid = " $3 " rép perso : " $6}' /etc/passwd**   
   Elle extrait les comptes d'uid supérieurs à 500 (grâce à $3>=500) et commençant par la lettre "j". En effet, l'appel à la fonction prédéfinie substr( $1, 0, 1) extrait une sous-chaine du nom ($1) à partir du 1er caractère (numéro 0) et sur une longueur 1, autrement dit l'initiale du nom.
6. On peut aussi sélectionner les enregistrements d'un fichier grâce à une expression rationnelle (ou régulière)   
   Par exemple la commande :  
   **gawk -F":" ' $1 ~/^[a-fj]/ { print NR $0 }' /etc/passwd**   
   sélectionne les lignes commençant par une lettre appartenant à l'intervalle de a à f, ou par j, et les affiche avec leur numéro de ligne. Attention ! l'opérateur [..] applique un OU implicite entre les lettres qu'il contient.

**RESEAUCONFIGURATION RESEAU**

Unix est un système qui intègre toutes les fonctionnalités réseaux reposant sur l'architecture **internet ARPA** (**Advanced Research Projects Agency**). Cette architecture permet ce communiquer avec d'autres ordinateurs sans tenir compte de la topologie du réseau et des systèmes d'exploitation des autres fournisseurs.

L'architecture **internet ARPA** se compose d'une série de protocoles structurés en couches. La base est le protocole **internet** de la couche réseau (**IP**). deux protocoles de la couche transport sont mis sur le protocoles **IP** : le protocoles **datagramme** utilisateur (**UDP**) et le protocole de contrôle de transmissions (**TCP**). Les protocoles de la couche présentation qui sont les précédents, sont la connexion à distance (**telnet**), le transfert de fichier (**ftp** et **tftp**) et enfin le transport de courrier (protocole **SMTP**, **Simple Mail Protocol**).

# I ) Configuration du réseau

un nom et une adresse doivent être attribués à chaque système du réseau. Le protocole **internet** est caractérisé par des adresses divisées en deux parties :

partie **adresse réseau** et partie **adresse locale**

Notre réseau local ROCAD (c'est-à-dire notre réseau universitaire) est un réseau de classe B ce qui explique que notre adresse est du type:

134.214.80.170 avec 134.214 adresse réseau

* 1. adresse locale

**1°) Le fichier de configuration de l’interface réseau**.

Les fichiers de configuration d'interface contrôlent le fonctionnement des interfaces logicielles associées aux périphériques réseau individuels. Lorsque votre système Red Hat Linux démarre, il utilise ces fichiers pour savoir quelles interfaces il doit afficher automatiquement et comment les configurer. Ces fichiers sont en général nommés ifcfg-*<nom>*, où *<nom>* se rapporte au nom du périphérique contrôlé par le fichier de configuration.

Le fichier ifcfg-eth0 constitue l'un des fichiers d'interface les plus communs; il contrôle la première *carte d'interface réseau* Ethernet ou *NIC* (de l'anglais 'Network Interface Card') dans le système. Dans un système comportant plusieurs cartes, il y aura plusieurs fichiers ifcfg-eth*<X>* (où *<X>* correspond à un numéro unique associé à une interface spécifique). Étant donné que chaque périphérique a son propre fichier de configuration, un administrateur peut contrôler sur le fonctionnement individuel de chaque interface.

Un fichier ifcfg-eth0 pour un système utilisant une adresse IP fixe ressemble à ce qui suit:

|  |
| --- |
| DEVICE=eth0  BOOTPROTO=none  ONBOOT=yes  NETWORK=10.0.1.0  NETMASK=255.255.255.0  IPADDR=10.0.1.27  USERCTL=no |

Les valeurs requises dans un fichier de configuration d'interface peuvent changer en fonction d'autres valeurs. Par exemple, le fichier ifcfg-eth0 pour une interface utilisant DHCP est légèrement différent, car les informations IP sont fournies par le serveur DHCP dans ce cas:

|  |
| --- |
| DEVICE=eth0  BOOTPROTO=dhcp  ONBOOT=yes |

Vous trouverez ci-dessous une liste de paramètres pouvant être configurés dans un fichier de configuration d'interface Ethernet:

* **BOOTPROTO**=*<protocole>*, où *<protocole>* correspond à l'une des valeurs suivantes:
  + none — Aucun protocole de démarrage ne devrait être utilisé.
  + bootp — Le protocole BOOTP devrait être utilisé.
  + dhcp — Le protocole DHCP devrait être utilisé.
* **BROADCAST**=*<adresse>*, où *<adresse>* correspond à l'adresse de diffusion.
* **DEVICE**=*<nom>*, où *<nom>* correspond au nom du périphérique physique (à l'exception des périphériques PPP à affectation dynamique où il s'agit du *nom logique*).
* **IPADDR**=*<adresse>*, où *<adresse>* correspond à l'adresse IP.
* **NETMASK**=*<masque>*, où *<masque>* correspond à la valeur de masque de réseau.
* **NETWORK**=*<adresse>*, où *<adresse>* correspond à l'adresse du réseau. Cette directive est déconseillée.
* **ONBOOT**=*<réponse>*, où *<réponse>* correspond à l'une des valeurs suivantes:
  + yes — Ce périphérique devrait être activé au démarrage.
  + no — Ce périphérique ne devrait pas être activé au démarrage.
* **USERCTL**=*<réponse>*, où *<réponse>* correspondant à l'une des valeurs suivantes:
  + - yes — Les utilisateurs autres que le super-utilisateur sont autorisés à contrôler ce périphérique.
    - no — Les utilisateurs autres que le super-utilisateur ne sont pas autorisés à contrôler ce périphérique.

## 

**2° ) le fichier /etc/hosts**

**/etc/hosts** contient une liste d'association nom de machine - adresse IP permettant de bypasser la résolution de nom classique (qui interroge les serveurs DNS) pour quelques serveurs dont on connaît l'adresse. Typiquement utilisé pour renseigner des noms de machines locales, plus facilement accessibles par leur nom qu'en tapant l'adresse complète.:

Chaque ligne de ce fichier contient les renseignements d'une machine réseau :

* l'adresse internet
* nom complet de la machine
* un ou plusieurs alias

|  |
| --- |
| 127.0.0.1 localhost.localdomain localhost  192.168.1.11 mamachine  192.168.1.12 monpc lepc |

**3°) le fichier /etc/services**

Le fichier **/etc/services** contient des informations sur les services connus et proposés par le réseau. Pour chaque service, le fichier **/etc/services** doit contenir une seule ligne de la forme :

**nom du service numéro de port / nom du protocole synonyme**

Le fichier /etc/services est utilisé par les démons, comme une liste de protocoles et des ports à mettre en œuvre lorsqu'un programme déterminé est appelé. Un port est un point final de communication sur une machine spécifique.

**4°) le fichier /etc/protocols**

Le fichier **/etc/protocols** contient des informations sur les protocoles connus qui sont utilisé par le réseau. Pour chaque protocole, le **fichier /etc/protocols** devrait contenir une seule ligne de la forme suivante :

**nom du protocole numéro de protocole synonyme**

Le fichier **/etc/protocols** est aussi utilisé par les démons pour déterminer le numétro de protocole à activer lorsqu'un protocole spécifique est appelé**.**

**5°) le fichier /etc/resolv.conf**

Ce fichier contient les renseignements de configurations du réseau

Les différentes directives de configurations sont :

**nameserver**  adresse IP du serveur de nom

**search**  le nom du domaine pour éviter de donner le nom qualifié d’une machine

**6°) le fichier /etc/sysconfig/network**

Ce fichier est utilisé pour spécifier les informations relatives à la configuration réseau. Les valeurs suivantes peuvent être utilisées :

* **NETWORKING=réponse**, les valeurs de **réponse** pouvant être : **yes** : lancement du réseau . **no** : pas de lancement du réseau
* **HOSTNAME=nom-machine**, **nom-machine** doit être le FQDN (Fully Qualified Domain Name), mais peut être le nom de machine de votre choix.
* **GATEWAY=gw-ip**, **gw-ip** étant l'adresse IP de la passerelle.

**II ) Les processus réseau**

Les outils suivants sont indispensables à connaître lorsque l'on utilise un système sous Linux. On ne peut ici donner toutes les options de ces commandes. N'oubliez donc pas que l'on peut avoir plus d'aide en tapant la commande suivie de --help, mais aussi man commande. Exemple **netstat --help** ou **man netstat**. Enfin souvenez vous que sous Linux, on ne peut utiliser indifféremment les majuscules et les minuscules (la commande ping existe, pas la commande Ping).

1. **ping**

Cette commande est normalement connue de tous. Elle existe dans tous les systèmes. Elle permet de vérifier si une machine distante répond. La syntaxe est des plus simple **ping -c 5 192.168.0.1** pour envoyer 5 pings à la machine dont l'adresse IP est 192.168.0.1.

On peut aussi utiliser le nom de la machine, si celle-ci est renseignée dans votre fichier Hosts ou dans un serveur DNS.On peut par exemple utiliser ping pour vérifier si la connexion est toujours active ou pour la monter.

Si vous ne placez pas l'option -c 5 pour n'envoyer que 5 pings, la commande ne s'arrête pas. Utilisez alors Ctrl C.

1. **ifconfig**

**ifconfig** permet de connaître la configuration de vos cartes réseau, mais aussi de changer celle-ci.

Pour changer la configuration de votre carte réseau, vous devez taper   
**ifconfig eth0 192.168.0.2 netmask 255.255.255.0 broadcast 192.168.0.255**Comme les valeurs que je viens de donner sont standards, vous pouviez simplement taper **ifconfig ETH0 192.168.0.2** (le netmask et broadcast proposés sont ceux correspondant à une adresse de classe C).    
Attention au redémarrage de la machine ce changement sera perdu. Il vous faut donc modifier en même temps le fichier /etc/sysconfig/network-script/ifcfg-eth0.   
Vous pouvez utiliser **linuxconf** pour faire plus simplement le même travail.  
On peut aussi désactiver une carte réseau **ifconfig eth0 down** et bien sûr la réactiver **ifconfig eth0 up**

1. **arp**

La commande arp permet de mettre en correspondance des adresses IP et les adresses MAC.

Les options possibles importantes sont   
**arp -a** pour avoir toutes les entrées ARP de la table  
**arp  -d nom\_de\_la\_machine** pour supprimer une entrée de la table  
**arp -s nom\_de\_la\_machine adresses\_mac**pour ajouter une nouvelle entrée dans la table.

1. **route**

Cette commande permet de voir, d'ajouter ou d'enlever les routes se trouvant déclarées sur votre machine. Ainsi pour indiquer à votre machine où aller trouver les adresses qui ne sont pas les adresses de votre réseau local, vous devez lui indiquer la passerelle (ou gateway) vers laquelle elle doit envoyer tous les paquets.  
Pour voir les routes indiquer **route -n** (on peut aussi utiliser **netstat -nr**) L'option -n permet de ne pas avoir la résolution des noms.

Pour ajouter une route par defaut : **route add default gateway 192.168.0.1** (La passerelle vers qui j'envoie tous les paquets qui ne sont pas pour le réseau local). Pour détruire cette route **route del default**

Pour ajouter une route vers une machine indiquer **route add -host 195.98.246.28 gateway 192.168.0.1** (Indiquer le netmask si celui-ci n'est pas un mask correspondant à la classe de votre adresse).

Pour ajouter une route vers un réseau indiquer **route add -net 195.98.246.0 netmask 255.255.0.0 gateway 192.168.0.1**

Enfin pour supprimer une de ces routes remplacer add par del.  
La gateway ou passerelle correspond la plupart du temps à votre routeur.  
Pour avoir la route que vous venez d'ajouter à chaque démarrage placer la commande dans le fichier /etc/rc.d/rc.local par exemple.

On peut aussi utiliser **linuxconf** pour faire la même chose.

1. **netstat**

Voilà une commande moins connue et pourtant très utile. Je ne peux ici commenter toutes les options, je vous propose de lire le **man netstat**. Elle permet en effet de connaître les ports en écoute sur votre machine,  sur quelles interfaces, avec quels protocoles de transport (TCP ou UDP), les connexions actives et de connaître les routes.   
Pour voir les connexions actives **netstat -nt**, pour les ports ouverts **netstat -ntl**.  
On peut aussi vérifier s'il existe une route par défaut, par exemple existe-t-il une route par défaut vers la machine 195.98.246.28 utilisez alors **netstat -nr | grep 195.98.246.28.**L'option -a énumère les ports en cours d'utilisation ou ceux qui sont écoutés par le serveur.  
L'option -i donne des informations sur les interfaces réseau.

1. **lsof**

**lsof** permet de lister les fichiers ouverts et les processus actifs.  
**lsof -i** indique les processus de type internet.

On peut ne demander que pour un protocole **lsof -ni tcp:25** ou que vers une machine **lsof -ni @192.168.0.1:25**

Pour connaître tous les fichiers ouverts par sur /hda1 utiliser **lsof /dev/hda1**.  
**lsof -i -a -p 1234** permet de connaître tous les ports réseau ouvert par le processus 1234 (-a est interprété comme AND).

**lsof -p 1234, 12345 -u 500, toto** permet de connaître tous les fichiers ouverts par l'utilisateur 500 ou toto ou par le processus 1234 ou 12345.  
Il existe des commandes pour faire cela (fuser,ps, netstat..), mais celle ci est très complète.

7 **traceroute**

traceroute permet de déterminer la route prise par un paquet pour atteindre la cible sur internet. On peut utiliser soit l'adresse IP, soit le nom d'hôte. Attention certains FireWall ou routeurs ne se laissent pas voir avec la commande traceroute.  
La commande traceroute est très utile pour savoir ou peut se trouver un blocage (plutôt ralentissement). Il existe un grand  nombre d'options, entre autre il est possible de choisir les gateway (jusqu'à 8) pour atteindre une machine. Je vous conseille donc encore une fois de lire le man traceroute.

 8 **telnet**

Telnet est l'outil indispensable à connaître. Il existe en tant que client sur tous les systèmes. Par contre Linux dispose en plus d'un serveur telnet permettant d'administrer à distance une machine (quoiqu'il existe maintenant un serveur telnet sur windows 2000). On peut ainsi administrer une machine linux depuis un Microsoft quelconque et, mais cela va de soi, depuis une autre machine linux.

En tant que client, telnet vous permet d'envoyer et de lire vos messages (voir ici).  
Pour pouvoir administrer à distance, il faut que le serveur telnet soit installé sur la machine que vous souhaitez administrer. Pensez aussi à vérifier que cela est autorisé dans le fichier etc/inetd.conf et dans /etc/hosts.allow et /etc/hosts.deny (voir tcp wrappers).   
Si vous devez vous en servir sur un réseau local ou sur internet, préférez lui **SSH** (ou la version autorisée en France **SSF**), car alors les mots de passe ne se promènent pas en clair sur le réseau.

Par défaut il n'est pas possible de se connecter en root avec une connexion telnet. Vous devez utiliser un autre compte et utiliser la commande su.

1. **ftp**

ftp est un outil qui permet de télécharger des fichiers entre machines. Vous connaissez les clients ftp comme ws\_ftp.

Sous Linux il existe un serveur ftp, que vous activez dans /etc/inetd.conf. Il est installé par défaut dans toutes les distributions. Ce serveur ftp n'est pas lié à l'installation d'apache, comme pour les systèmes Microsoft où vous devez installer IIS pour bénéficier de ce service. Attention toutefois le serveur ftp pose un problème de sécurité important, utilisez plutôt **SFTP**, qui est disponible avec **SSH**.

Vous disposez aussi d'un client ftp en ligne de commande sous Linux comme sous Microsoft. La syntaxe étant pratiquement la même.

Voici les commandes que vous allez utiliser le plus :

|  |  |
| --- | --- |
| **dir** | pour lister un répertoire |
| **cd nom\_du\_répertoire** | pour changer de répertoire |
| **get mon\_fichier** | pour copier un fichier vers votre client (obtenir). Il se place alors dans le répertoire où vous vous trouviez. |
| **mget \*** | copier tous les fichiers du répertoire vers votre station |
| **put mon\_fichier** | pour copier un fichier vers le serveur |
| **mput \*** | pour copier les fichiers se trouvant dans votre répertoire. |
| **binary** | pour copier en mode binaire. |
| **exit** | pour quitter |

Il existe un grand nombre d'autres commandes. Mais vous avez là les principales, pour copier des fichiers entre machines. La commande ftp vous rendra un grand nombre de services, car elle permet assez simplement d'échanger des fichiers entre linux et windows, sans avoir à installer un client ftp ou à configurer samba.

1. **nslookup**

L'utilitaire nslookup permet d'interroger un serveur de nom (serveur dns) afin d'avoir des informations sur un domaine ou sur une machine. Par défaut nslookup utilise le serveur de nom configuré sur votre machine, vous pouvez toutefois interroger un autre serveur de nom.   
[root@aleu /]#nslookup  
Default Serveur: localhost        Car j'ai un serveur dns sur ma machine   
Address: 127.0.0.1

>help     Pour avoir de l'aide

>**set type = MX**     Pour lister les entrées de type MX (à savoir les serveurs SMTP du domaine).

>ac-creteil.fr    Le nom du domaine dont vous voulez avoir des MX

Remplacer MX par le type d'enregistrement que vous souhaitez avoir. Par exemple NS pour les serveurs de nom d'un domaine, SOA  pour start of authority, PTR pour le reverse, A pour une machine.

Pour avoir toutes les informations

**set type=ANY** puis le nom du domaine.

On peut aussi utiliser la commande **ls -t CNAME nom\_du\_domaine** pour avoir tous les enregistrements de type cname (les alias).

Pour interroger un autre serveur DNS que votre serveur par défaut **server NAME 195.98.246.50**.

1. **host**

Commande presque équivalente, mais dont l’usage est plus simple.  
**host –a wanadoo.fr 193.252.19.1** Donne de informations sur le domaine wanadoo.fr  
**host –v –t mx wanadoo.fr 193.252.19.1** Donne des informations sur les MX du domaine Wanadoo (vous pouvez indiquer un autre DNS).  
**host –l –t any wanadoo.fr** pour obtenir toutes les machines du domaine Wanadoo. Ici je n’ai pas indiqué de DNS. Le DNS par défaut sera utilisé, mais si ce serveur DNS n’est pas le DNS du domaine il indiquera qu’il n’est pas authoritative.

1. **who**

Cette commande permet de connaître les personnes qui sont loguées sur votre machine.

1. **last**

Cette commande vous permet de voir les dernières connexions ayant eu lieu sur votre machine (en fait il lit le fichier /var/log/wtmp).

**last** sans rien, vous affiche toutes les informations.  
**last philippe** toutes les connexions de l'utilisateur philippe.  
**last reboot** tous les reboot de la machine avec la date.   
**lastb** est une variante de last, dans la mesure ou il ne cherche que les mauvais login (il lit le fichier /var/log/btmp)

**14 finger**

finger est un service qui vous permet d'obtenir des informations sur les comptes utilisateurs de votre machine. Ce service est à proscrire. sa syntaxe est toutefois assez simple **finger toto@la\_machine\_distante**  
Pour avoir plus d'informations utiliser l'option -l   
Cet exemple montre les deux connexions ouvertes sur la machine aleu.  
  
[root@aleu philippe]# finger -l  
Login: root Name: root  
Directory: /root Shell: /bin/bash  
On since Sun Oct 22 18:34 (CEST) on tty1 39 seconds idle  
(messages off)  
No mail.  
No Plan.  
  
Login: philippe Name: philippe  
Directory: /home/philippe Shell: /bin/bash  
On since Sun Oct 22 18:48 (CEST) on pts/0 from 10.100.1.19  
No mail.  
No Plan.   
  
Par défaut et par sécurité les machines distantes ne permettent pas, ou plus les commandes finger.

**tcpdump**

tcpdump permet de faire des captures de paquets sur votre réseau. Je présente ici tcpdump car on le trouve sur tous les cd des distributions. Il n'est pas le plus agréable à utiliser et des utilitaires de ce type plus conviviaux existent. Mon but n'est pas ici de vous montrer comment devenir un pirate (en capturant les mots de passe qui circulent en clair sur votre réseau), mais plus de vous permettre de vérifier par exemple lorsque votre routeur monte la ligne sans que vous ne soyez capable de donner l'origine de cette montée de ligne. Comme de plus cela arrive la nuit (toujours quand on n'est pas là..!), il peut être utile de placer tcpdump et de capturer les  paquets à destination de votre routeur et uniquement cela. Au petit matin en analysant le résultat vous savez quelle machine et quel protocole monte la ligne.Par exemple pour intercepter tous les paquets vers la machine 10.100.1.5 sur le port telnet  
**tcpdump -l -q -x host 10.100.1.5 and port telnet**  
Pour intercepter tous les paquets d'une machine vers une autre sur le port telnet  
**tcpdump -l -q -x dst 10.100.1.5 and src 10.100.1.19 and port telnet and tcp**Pour avoir tous les paquets qui arrivent sur votre machine 10.100.1.5 ne pas indiquer la source.

1. **nmap**

nmap  est un outil pour scanner les ports ouverts sur une machine distante. Son utilisation est des plus simple **nmap 192.168.0.1** pour scanner une machine ou **nmap 192.168.0.0/24** pour scanner les machines se trouvant dans le plan d'adressage 192.168.0.0/24  
Utilisez l'option -v pour avoir plus d'informations. On peut bien sur scanner que certains protocoles, par défaut le protocole scanné est TCP. Pour scanner les deux **nmap -v -sU -sT 192.168.0.1**  
  
Les options disponibles sont :  
  
-sT  Scanne les ports TCP -sU pour UDP (attention cela est inscrit dans les fichiers de log de la machine cible).  
-sS  Est identique au précédent sauf que cela ne laisse pas de trace. (il y a une différence quant à la méthode mais cela n'est pas l'objet de cette présentation).  
-sP En fait un ping.  
-p 20-140 Ne scanne que les ports entre 20 et 140, 1024- scanne tous les ports à partir de 1024.  
-I Pour avoir plus d'information sur le port ouvert, ne marche que si identd n’est pas utilisé sur la machine cible.  
-O Permet de connaître l’os (voir aussi –osscan\_guess).   
-P0 Permet de scanner les machines qui n’autorisent pas les ICMP echo request.  
    
[root@aleu philippe]# **nmap -v -sU -sT 10.100.1.1**  
Starting nmap V. 2.30BETA17 by fyodor@insecure.org ( www.insecure.org/nmap/ )  
Host (10.100.1.1) appears to be up ... good.  
Initiating TCP connect() scan against (10.100.1.1)  
Adding TCP port 139 (state Open).  
Adding TCP port 25 (state Open).  
Adding TCP port 110 (state Open).  
Adding TCP port 21 (state Open).  
Adding TCP port 135 (state Open).  
Adding TCP port 80 (state Open).  
Adding TCP port 1026 (state Open).  
The TCP connect scan took 1 seconds to scan 1534 ports.  
Initiating FIN,NULL, UDP, or Xmas stealth scan against (10.100.1.1)  
The UDP or stealth FIN/NULL/XMAS scan took 4 seconds to scan 1534 ports.  
Interesting ports on (10.100.1.1):  
Port State Service  
21/tcp open ftp  
25/tcp open smtp  
80/tcp open http  
110/tcp open pop-3  
135/tcp open loc-srv  
137/udp open netbios-ns  
138/udp open netbios-dgm  
139/tcp open netbios-ssn  
1026/tcp open nterm   
Nmap run completed -- 1 IP address (1 host up) scanned in 5 seconds   
  
Il n'est pas installé par défaut sur votre machine, mais on commence à le trouver sur le CD d'installation des distributions. Il existe une interface graphique (installer nmap-frontend en version rpm).  
Evitez de scanner des machines qui ne sont pas vos machines. Ce produit a pour vocation de vérifier si votre machine est correctement configurée, pas pour tester  les autres.

1. **ntop**

Un outil indispensable. Il tourne comme daemon et vous pouvez voir les résultats (entre autre ) via une interface web par défaut sur le port 3000 qu'il est possible de changer avec l'option "**-w**".

**NFS (Network File System)**

## Généralités

* Il s'agit du protocole standard de partage réseau entre machines Unix, créé par SUN vers 1980.  
  Il comprend l'ajout de fonctionnalités supplémentaires (dans la couche session au dessus de TCP/IP), les **RPC** =(*Remote Procedure Calls*)
* Donc une machine joue le rôle de serveur de fichiers. Elle est appelée **serveur NFS**, et
  + on dit qu'elle **exporte** tout (arborescence racine /) ou partie de son système de fichiers,
  + en le partageant par une liste de stations accessibles par réseau,
  + en installant toutefois des restrictions d'accès.
* Comme toute ressource extérieure doit être intégrée dans le système de fichiers Linux, cet accès ne pourra être permis qu'à l'aide d'un processus de montage : une partie de l'arborescence d'une machine Linux "serveur", est exportée ce qui lui permet d'être intégré dans le système de fichiers d'une machine Linux "cliente".
* L'utilisateur peut monter cette arborescence exportée par le serveur, sur un point de montage, de façon tout-à-fait semblable au [montage de systèmes de fichiers](http://www.ac-creteil.fr/infolyc/linux/formation/systemes-fichiers.html#fstab) des divers périphériques.  
  Le montage peut s'effectuer en cours de session de travail par la commande interactive **mount**.
* Mais dans un cadre de travail stable, où le serveur est dédié, il est souhaitable de monter la ressources **NFS** au démarrage.  
  Il suffit pour cela d'inclure la description du montage sur une ligne de **/etc/fstab**. On peut comparer le processus à la "connexion à un lecteur réseau" sur d'autres systèmes.
* Dès lors, pour l'utilisateur sur la machine cliente, la ressource est accessible comme si elle résidait sur un périphérique local.

**NFS** (Network File System), un système de fichiers par réseau, est un service très répandu utilisant **RPC** dans le monde Unix. Il donne accès à des fichiers partagés sur des machines distantes. L'utilisateur les voient alors comme s'ils étaient stockés sur son PC.  
L'avantage de **NFS**, est de pouvoir (par exemple) stocker tous les répertoires utilisateurs sur un serveur, et de le monter à chaque démarrage sur les stations de travail: ainsi, les utilisateurs possédant un compte sur des machines distinctes pourront travailler sur les mêmes fichiers.

Pour utiliser NFS, votre noyau doit supporter le système de fichier **NFS** ( compilé dans le noyau ou en tant que module)

Pour monter un répertoire, il suffit de taper:

**mount machine:/repertoire\_distant /repertoire\_local**

La commande **mount** tentera alors de se connecter au daemon ***mountd*** de *machine*. Le serveur teste si la machine locale est autorisée à accéder au répertoire distant, et si c'est le cas, le répertoire sera monté en local sous le nom */****repertoire\_local***.

Il est important que les numéros et groupe des utilisateurs des 2 machines soit les mêmes. On peut pour cela utiliser **NIS**.   
Si vous voulez devenir un serveur **NFS**, vous aurez besoin du démon ***nfsd*** et ***mountd***. Ce sont des services **RPC**: ils devront être lancés après le démon ***portmap*** dont vous aurez besoin.

Toute la configuration de **NFS** se trouve dans le fichier ***/etc/exports***, qui répertorie les répertoires partagés.

Voilà un exemple de fichier /etc/*exports*:

# Fichier exports de quark.nunux.fr  
/home       mulder(rw) scully(rw)   
/usr/TeX       (ro)

Chacune des lignes définit un répertoire et les hôtes autorisés à y accéder. Les noms des machines peuvent contenir les caractères génériques \* et ? mais ce n'est pas conseillé pour des raisons de sécurité. Si aucun nom n'est spécifié, alors n'importe qui pourra le monter sur son PC par **NFS**.

Le nom de chaque machine est suivi d'une liste facultative de drapeaux (flags), séparés par des virgules et mit entre parenthèses. Il peuvent prendre les valeurs suivantes:

***secur****e*: Réclame que les requêtes proviennent d'un port internet de numéro inférieur à 1024. Elle est en service par défaut; insérer ***insecure*** pour la désactiver.

***ro***: N'autorise que des requêtes en lecture-seule sur ce volume **NFS**.

***link\_relative***: Convertit les liens symboliques absolus (dont le contenu commencent par un slash) en lien reatifs ajoutant le nombre nécessaire de "../" pour aller de l'emplacement du lien à la racine du serveur. Cette option peut réservé parfois quelques surprises si la hiérarchie n'est pas montée à la racine.

***root\_squash***: Transforme les requêtes UID/GID 0 en UID/GID anonyme. Ceci ne s'applique pas aux autres UID/GID sensible comme bin.

***no\_root\_squash*** Contraire de root\_squash. Cette option est utile pour les stations sans disques.

***squash\_uids*** et ***squash\_gid***: Cette option permet de préciser une liste d'UID et de GID qui seront convertis en utilisateurs anonymes. Elle se présente ainsi: squash\_uids=0,15,25-50

***all\_squash***: Convertit tous les UID/GID en utilisateurs anonymes.

***map\_daemon***: Cette option active la conversion dynamique des **UID/GID**. Chaque **UID** d'une requête **NFS** sera transformé en **UID** équivalent sur le serveur, et l'**UID** des réponses **NFS** subira la transformation inverse. Ceci nécessite que le démon **ugidd** soit en fonctionnement sur le client. La configuration par défaut est **map\_identity**, qui laisse les **UID**s intacts. Les options de `squash' s'appliquent que la conversion dynamique soit utilisée ou non

## Installation

#### Sur le serveur NFS

* Tout d'abord, les services **portmap** qui gère les connexions RPC, et **nfs** ont dû être installés initialement, sinon le faire   
  Vérifier avec l'utilitaire **ntsysv** que les services **portmap** et **nfs** sont bien activés automatiquement au démarrage.

Pour vérifier que les processus correspondant sont en exécution :

* ps aux | grep portmap
* Sur le modèle de gestion des services, on peut faire appel aux scripts de gestion placés dans */etc/rc.d/init.d/*, pour lancer (option start), arrêter (stop), connaitre la situation (status), relancer (restart) ... Voici pour lancer NFS (sur une distribution de type RedHat)
* # /etc/rc.d/init.d/portmap start
* Starting portmapper: [OK]
* # /etc/rc.d/init.d/nfs start
* Starting NFS services: [OK]
* Starting NFS quotas: [OK]
* Starting NFS deamon: [OK]
* Starting NFS mountd: [OK]
* Le fichier **/etc/exports.**  
  Ce fichier (à créer s'il est absent) contient la liste des exportations.   
  Sur chaque ligne, on précise un répertoire du système de fichiers, suivi par la liste des  machines distantes clientes autorisées à les monter. Si cette liste est vide, toutes les stations accessibles sont autorisées.   
  Exemples d'exportation déclarées dans le fichier **/etc/exports** sur le serveur p00
* # repertoire liste-machines (liste-options)
* /home/jean pc2(ro) pc3(rw)
* /usr/bin pc2(ro) pc3(ro)
* /var/www/html \*.fctice.ac-creteil.fr (ro) pc3 (rw)
* /usr/share/doc (ro)

Pour valider un changement opéré dans ce fichier de configuration, faire appel à la commande :

# exportfs –a

#### Sur la station cliente

* On crée un rép. de montage, situé pour des tests, dans **/mnt**, par exemple sur la machine pc3 :  
  [root@pc3 /]# **mkdir /mnt/nfs**
* Puis on effectue le montage, sur le point de montage précédent, de la ressource /home/httpd/html (qui a été exportée par **p00** [root@pc3 /]# **mount -t nfs p00:/home/httpd/html /mnt/nfs** L'utilisateur sur pc3 pourra alors mettre à jour le site WEB distant sur p00
* Syntaxe  générale  
  **mount -t nfs nom-machine:arborescence point-montage**
* Respects par nfs des droits   
  Bien sûr les permissions des fichiers importés s'appliquent vis à vis de l'utilisateur, notamment en ce qui concerne la directive **(rw)**. On ne pourra mettre à jour sur la station cliente, un fichier exporté que s'il posséde la permission **w** vis-à-vis de l'utilisateur.
* Automatisation du montage  
  Pour cela, il suffit d'ajouter le contenu de la commande précédente dans une ligne du fichier [/etc/fstab](http://www.ac-creteil.fr/infolyc/linux/formation/systemes-fichiers.html#fstab)

**p01:/home/httpd  /mnt/import  nfs  auto, user**

* Autres paramètres de montage  
  **insecure** autorise les accès non authentifiés

### Exemple de mise en oeuvre

 Ce que veut avant tout un utilisateur, c'est pouvoir travailler dans son répertoire personnel situé sur du serveur, à partir d'une station Linux quelconque.

 Il doit se connecter et s'authentifier sur le serveur NIS qui tient à jour la base de données des comptes utilisateurs

* Sur le client pc1 on monte la ressource **/home** exportée sur le serveur sur **/home** local, au démarrage ou bien en root.  
  Ce montage ca masquer les comptes éventuellement présents sur le client   
  **# mount -t nfs serveur:/home /home**
* Dès lors, l'utilisateur jean accède à ses fichiers par :  
  **[jean@pc1 ] cd /home/jean**

### 